jrg:
nr:

JUNI 1983

++

- ACORN Nieuws -

++

De grote doorbraak.

(AL WEER?)

zie blz.4



++

ACORN NIEUWS Uitg.Fed.Acorn Computerclubs Ned/B.
Verschijnt 6-8 x p/jr.

UIT DE FEDERATIE:

PRINTEN: Nog beschikbaar: CMOS-geheugenprinten ƒ 35,-
Club-SCHAKELprinten ƒ 45,-
incl. 74LS133
EPROM-programmer printen ƒ 17,-

Bestellen: Giro 5244293, t.n.v. Acorn Computerclub, te Malden. Zo mogelijk via uw Penningmeester.

MEDEDELING voor de regionale bandjes archieven:

Dinsdag 24 mei jl. is Hans Kuiper bij een ongeluk om het leven gekomen. Hans was de auteur van het programma FAX. In dit programma staat voor meer informatie zijn adres en telefoonnummer. Wilt u s.v.p. dit adres en telefoonnummer vervangen door dat van ondergetekende?

Nico Stad
Plataanweg 47
1544 PB ZAANDIJK
075-280808

REDACTIE-adres: Ton Otten, J.A. de Gravenlaan 17, 2381 TA Zoeterwoude.

DEAD-LINE A.N. 4, 1983: 15 juli 1983.

AUTEURS: Zie pagina 29!

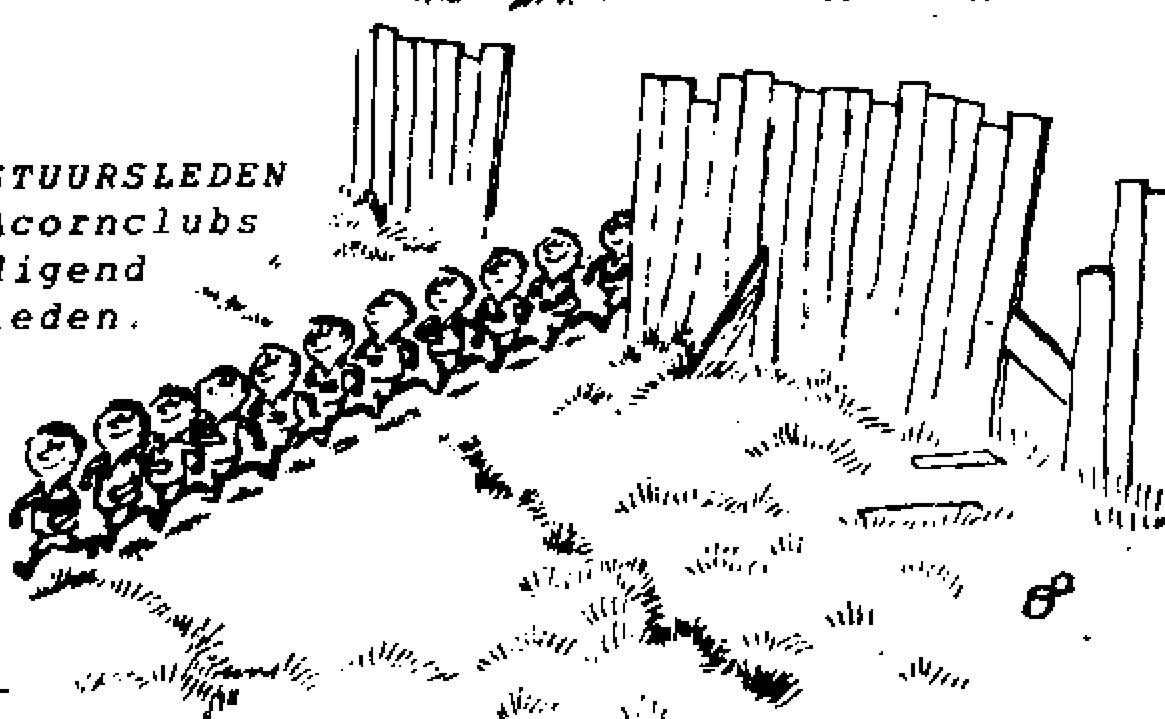
IN DIT NUMMER:

.--++ooOoOoOoOoOoOoo++--.

28 MEI 1983 :



Eerste ALGEMENE LEDENVERGADERING



Kort VERSLAG:

Aanwezig waren de BESTUURSLEDEN van 14 [van de 15] Acornclubs tesamen vertegenwoordigend 642 [van de nu 665] leden.

Afwezig z.k. Zeeland.

Behandeld en met alg. stemmen goedgekeurd de verbeterde

FEDERATIEVE STATUTEN.

Idem: Het financieel jaarverslag van 1982 [tekort f 358.-] en de begroting van 1983 [geschat batig saldo [reserve] van f 2000.-]

Met eveneens algemene stemmen werd volgens Bestuursvoordracht het nieuwe BESTUUR gekozen, te weten:

Voorzitter: ARNO MILLENAAR

Secretaris: G.H.BORGHAERTS

Pen.meester: GERHARD VISSER

De Voorzitter van de A.C.Belgie peilde vervolgens de interesse voor een F E E S T W E E K E N D in Rendeux-Haut [Belgische Ardennen] op 26/27 November '83 dit ter gelegenheid van de eerste Verjaardag van de A.C. Belgie ! ! !
Voor nadere gegevens: VRAAG UW BESTUUR !

Parallel aan de Algemene Vergadering vonden vergaderingen plaats van de diverse "netwerken", t.w. REDACTIE , HARDWARE, SOFTWARE, ARCHIEVEN. Alles tesamen zo'n 60 Acornuiten.

Vooral HARDWARE had de smaak tepakken van bitterballen en van appelmoes en ging welgemoed door tot middernacht !

Alles tesamen: Een zeer geslaagde contactdag. Alle lof voor de opkomst, voor de prima sfeer en voor de gastvrouw Marion Gruijters,scheidend penningmeesteres.

G.Borghaerts,Secretaris.

OF: WAT TE DOEN ALS SNELHEID BELANGRIJK WORDT.

a) Gebruik afkortingen zoveel als maar mogelijk is. Elke letter die teveel in het programma staat kost tijd (en geheugenruimte)

b) Een puntkomma teveel of een spatie aan het einde van een regel kost zéér veel tijd. De interpretator gaat dan nl. op zoek naar een statement. Uiteraard is er niet zo'n statement in de BASIC en dus wordt er verder gezocht in de floating-point ROM. Ook hier bestaat geen statement 'puntkomma' of 'spatie' en dus wordt er ook nog eens verder gezocht in de utility-ROM. Pas als de computer ook hier niets vindt, komt hij tot de conclusie dat het misschien wel een puntkomma of spatie zou kunnen wezen. Al met al een aanzienlijk tijdverlies.
Extra puntkomma's zijn gemakkelijk te voorkomen. Extra spaties aan het eind van een regel willen nog wel eens ontstaan doordat er met de COPY-toets net even iets te ver wordt gecopieerd en deze extra spatie dan niet wordt weggehaald met DELETE. Deze extra spaties zijn later moeilijk terug te vinden. Dus: niet te ver gaan met de copytoets!!!

c) Gebruik variabelen in plaats van constanten!

```
 0 REM TIMING
10 FOR I=1 TO 5000
20 K=K-5;L=L-222
30 NEXT
40 END

   18.8 sec
```

```
 0 REM TIMING
 5 C=5;D=222
10 FOR I=1 TO 5000
20 K=K-C;L=L-D
30 NEXT
40 END

   15.0 sec
```

Het verschil wordt veroorzaakt doordat de computer bij een constante elke keer de karakters om moet zetten in een getal, terwijl bij een variabele het getal al gewoon in z'n geheugen staat.

d) Gebruik labels in GOTO en GOSUB.

e) Denk goed na over het probleem !!!!

Bv.
```
FOR J=1 TO 10
FOR K=1 TO 10
AA(K)=AA(J)x2-K
NEXT K
NEXT J
```
is langzamer dan:
```
FOR J=1 TO 10
C=AA(J)x2
FOR K=1 TO 10
AA(K)=C-K
NEXT
NEXT
```

f) Gebruik assembler voor die stukken van het programma die zeer vaak worden uitgevoerd. Machine-taal kan wel tot een factor honderd sneller zijn dan BASIC hoewel een machine-taal programma meestal langer is dan een gelijkwaardig BASIC-programma.

g) Gebruik tabellen of array's ipv. functies. Als we bv. steeds een sinus willen uitrekenen terwijl het geheugengebruik en de nauwkeurigheid niet zo van belang zijn, dan kan het handig zijn als je eerst een sinustabel maakt, en dan de waarde van de sinus gewoon opzoekt in de tabel.

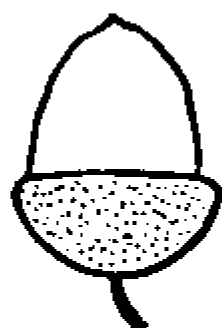
het programma 'cirkels' is een op snelheid geschreven programma. Er wordt gebruik gemaakt van een sinus- annex cosinus-tabel en van een vermenigvuldigingstabel. De vermenigvuldiging gebeurt in slechts 81 klokcycli (81 microsec) en is op 7 bits nauwkeurig.
Het programma kan ongeveer elf cirkels per seconde tekenen in mode 4, waarbij elke cirkel bestaat uit 256 punten.
Om 256 cirkels te tekenen zijn dan:
65536 sinus berekeningen,
65536 cosinus berekeningen,
65536 plot instructies,
131072 vermenigvuldigingen en slechts
23 seconden nodig.

De vermenigvuldiging die ik hiervoor gebruikt heb is (naar ik aanneem) nogal bijzonder en behoeft wellicht enige toelichting. Een byte is opgebouwd uit twee helften, die gewoonlijk 'nibbles' (spreek uit: nibbels) worden genoemd. Het byte #F4 is dus opgebouwd uit F en 4.
De vermenigvuldigingstabel bevat alle 256 vermenigvuldigingen van het linker- maal het rechter nibble.
Dus als de tabel begint op bv. #3A00 en we willen het antwoord weten van 6x5 dan hoeven we slechts te kijken op de plaats #3A65 waar dan het antwoord staat.
Om nu het hoge byte van het antwoord van bv. #45xA9 te bepalen zoeken we op in de tabel 5xA, 9x4 en 4xA. vervolgens tellen we op 5xA en 9x4 en delen dit door 16 (LSR A), hierbij tellen we nog 4xA op en voilà.
De eerste 7 bits zijn altijd correct, terwijl het laatste (nulde) bit meestal goed is. Het gaat nl. mis als er van 5x9 een 'carry' zou ontstaan. In principe kan dit gemakkelijk worden opgelost door ook 5x9 in de tabel op te zoeken, maar voor het programma 'cirkels' is 7 bits nauwkeurig genoeg (behalve voor kleine cirkels).

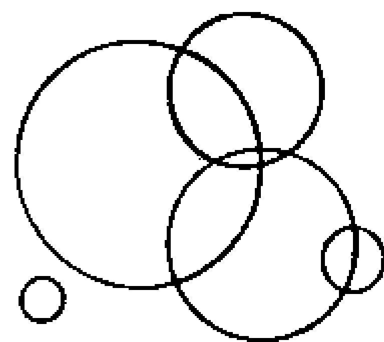
Om nog grotere executie snelheid te verkrijgen heb ik niet gebruik gemaakt van de BASIC plot-instructies, maar van een machine-taal subroutine (uit de BASIC-ROM) die een puntje op het scherm zet. Daarbij moet #5E gelijk aan 1 zijn (set-point) en bevat #5A de x-coordinaat en #5C de y-coordinaat van het te plotten punt. Een JSR #F7B2 zorgt er in dit geval dan voor dat het puntje inderdaad geplot wordt.

::

```
LIST
   0 REM CIRCLE
   5 DIMLL2;?#23=0;?#24=?#24+1
  10 DIMT255,S(255+64)
  20 C=S+64
  30 DIMP-1
  40[\multiply
  50\ AB*CD
  60:LL0;STA #80;AND @#F0;STA #82 (A)
  70;LDA #81;AND @#0F (D)
  80ORA #82 (A);TAX
```



```
 90LDA #81;AND @#F0;STA #84 (C)
100LDA #80;AND @#0F (B)
110ORA #84 (C);TAY
120SEC
130LDA T,X;ADC T,Y;ROR A;LSR A;LSR A;LSR A;TAY
140LDA #84 (C);LSR A;LSR A;LSR A;LSR A
150ORA #82 (A);TAX
160CLC;TYA;ADC T,X;RTS;
170\main
180:LL2;LDA @1;STA #5E
190:LL1;LDX #86;LDA S,X sin;STA #81;LDA #87;JSR LL0 mul
200ADC #91;STA #5C
210LDX #86;LDA C,X cos;STA #81;LDA #87;JSR LL0 mul
220ADC #90;STA #5A;JSR #F7B2
230DEC #86;BNE LL1;DEC #87;BNE LL1;RTS;]
240
250 REM Multiply-tabel
260
270 P."IK MAAK NU DE VERMENIGVULDIGINGSTABEL"'
280 K=0
290 F.I=0TO15;F.J=0TO 15
300 K?T=I*J;K=K+1
310 N.;N.
320
330 REM Sinus-tabel
340
350 P.''"OVER DE (CO)SINUS-TABEL DOE IK  EVEN WAT LANGER"''
360 F.I=0TO255+64
370 I?S=%(SIN(I/128*PI)*128+127.5)
380 N.;Q=LL2;G=#90
390 P."IK BEGIN MET TEKENEN ALS JE OP  EEN TOETS DRUKT."
400 P." (STAAT DE STOP-WATCH KLAAR)"'
410 LINK#FFE3
420 CLEAR4;!G=0;?#86=0;?#87=0;LI.Q;P.$7
430 F.I=1TO300;WAIT;N.;P.$12"ZO DAT WAREN 256 CIRKELS."'''
440 P."EN DAN NU..."
450
460 REM Sinus met Floating-Point
470
480 ?(P-5)=#60;F.I=1TO300;WAIT;N.
490 CLEAR4;MOVE0,92;F.X=0TO255;DRAWX,%(SIN(X/128*PI)*64+92);N.
500
510 REM Sinus met sinus-tabel
520
530 F.I=0TO300;WAIT;N.
540 CLEAR4;MOVE0,92;F.X=0TO255;DRAWX,(S?X/2+30);N.
550 F.I=0TO300;WAIT;N.
560
570 REM Random cirkels
580
590 CLEAR4;W=#87;DO!G=R.;?W=R.;LI.Q;U.0
600
610 Frans van Hoesel
```



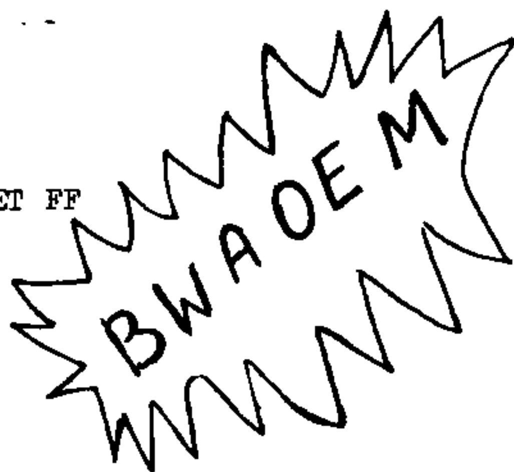
```
)L.
   0 REM FAST OUTPUT
  10 REM routine gebruikt 117 bytes
  20 REM de plaats van de routine
  30 REM wordt bepaalt door:
  40 REM P op regel 50
  50 P=#2800;COPY#FE52,#FE5C,P
  60 P!11=P+16;P!13=#FE5F4C
  70 RELOC#FCEA,#FD4F,P+16
  80 P!#58=#EAEAEA60
  90 P!#6A=#EAEAEA48
 100 ?#208=P%256;?#209=P/256
 110 END
 120
 130 het nadeel van deze routine
 140 is, dat het printen niet meer
 150 noise-vrij gebeurt,maar in de
 160 praktijk valt dit erg mee.
 170
 180 Klaas de Raad
```



Het volgende programmaatje van Wim Schreuder zet bliksemsnel het hele geheugen tussen BEGINADRES en EINDADRES vol met ≠FF.

Het kan gebruikt worden ter beveiliging van een bepaald programma tegen onbevoegd RUNnen, bijvoorbeeld door in het te beveiligen programma een 'zelfmoordregel' op te nemen. Deze kijkt naar een bepaalde byte; bevat deze niet de afgesproken, geheime code, dan blaast het programma zichzelf op!

```
 10 DIM LL3
 20 FOR I=1 TO 2
 30 P=≠2800
 40 P.$21
 50[
 60:LL3 LDX  ≠80        MSB BEGINADRES
 70     STX  ≠81
 80     LDA@≠00         LSB BEGINADRES
 90     STA  ≠80
100     LDY@≠00
110     LDA@≠FF         GEHEUGEN VOL MET FF
120:LLO STA(≠80),Y
130     INY
140     BEQ  LL1
150     JMP  110
160:LL1 INC  ≠81
170     INX
180     CPX@≠98         MSB EINDADRES
190     BNE  LLO
200     RTS
210]
220 NEXT I;P.$6;LINK LL3;END
```



"BENCHMARKING"

(naar aanleiding van een artikel in Micro Choice winter 1982)

Om computers onderling te vergelijken worden vaak zogenaamde benchmarks gebruikt. Een benchmark is een set standaard testen die een indruk geven van de efficiëntie van machine.

De in de microcomputerwereld meest gebruikte testen zijn die welke in 1977 door Kilobaud Magazine in Amerika werden geïntroduceerd. Het gaat hierbij om acht simpele programmas waarvoor nauwkeurig de looptijd moet worden gemeten. Je kunt dit doen door voor elk programma meerdere keren te meten en dan het gemiddelde te nemen. (Ik heb de Real Timer van "Acorn Nieuws Noord" januari op 60 Hz. laten lopen. En de Atom zelf laten meten.)
Als je de resultaten hieronder bekijkt moet je ook bedenken dat sommige van de andere machines zelfs op 2 MHz. lopen of twee processors hebben. In een geval is er zelfs sprake van een 16 bits machine!

1
```
1ØØ PRINT"S"
2ØØ FOR K=1 TO 1ØØØ
3ØØ NEXTK
5ØØ PRINT"E"
6ØØ END
```

2
```
1ØØ PRINT"S"
2ØØ K=Ø
3ØØ K=K+1
4ØØ IF K<1ØØØ THEN3ØØ
5ØØ PRINT"E"
6ØØ END
```

3
```
1ØØ PRINT"S"
2ØØ K=Ø
3ØØ K=K+1
31Ø A=K/K*K+K-K
4ØØ IF K<1ØØØ THEN3ØØ
5ØØ PRINT"E"
6ØØ END
```

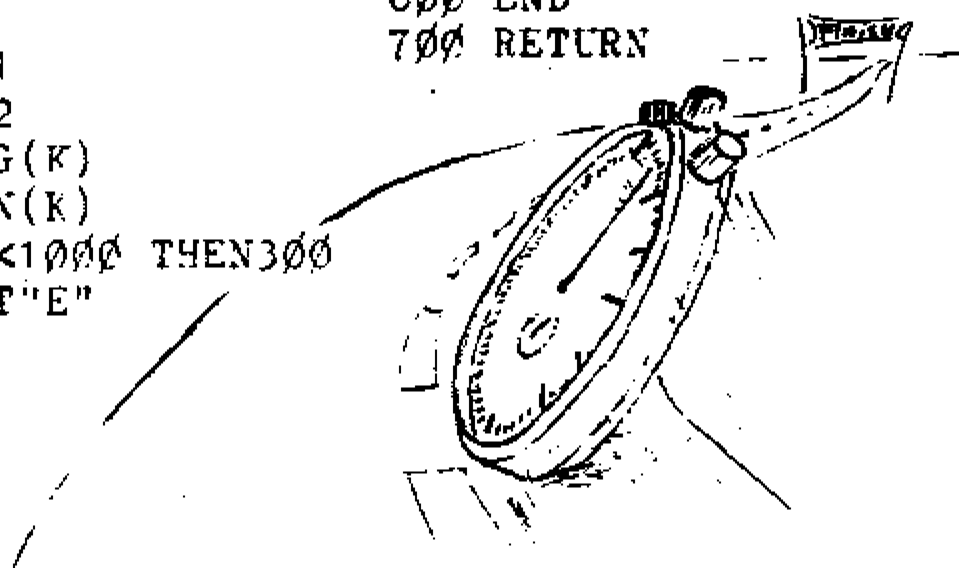
4
```
1ØØ PRINT"S"
2ØØ K=Ø
3ØØ K=K+1
31Ø A=K/2*3+4-5
4ØØ IF K<1ØØØ THEN3ØØ
5ØØ PRINT"E"
6ØØ END
```

5
```
1ØØ PRINT"S"
2ØØ K=Ø
3ØØ K=K+1
31Ø A=K/2*3+4-5
32Ø GOSUB7ØØ
4ØØ IF K<1ØØØ THEN3ØØ
5ØØ PRINT"E"
6ØØ END
7ØØ RETURN
```

6
```
1ØØ PRINT"S"
2ØØ K=Ø
25Ø DIM M(5)
3ØØ K=K+1
31Ø A=K/2*3+4-5
32Ø GOSUB7ØØ
33Ø FOR L=1 TO 5
34Ø NEXTL
4ØØ IF K<1ØØØ THEN3ØØ
5ØØ PRINT"E"
6ØØ END
7ØØ RETURN
```

7
```
1ØØ PRINT"S"
2ØØ K=Ø
25Ø DIM M(5)
3ØØ K=K+1
31Ø A=K/2*3+4-5
32Ø GOSUB7ØØ
33Ø FOR L=1 TO 5
335 M(L)=A
34Ø NEXTL
4ØØ IF K<1ØØØ THEN3ØØ
5ØØ PRINT"E"
6ØØ END
7ØØ RETURN
```

8
```
1ØØ PRINT"S"
2ØØ K=Ø
3ØØ K=K+1
33Ø A=K^2
34Ø B=LOG(K)
35Ø C=SIN(K)
4ØØ IF K<1ØØØ THEN3ØØ
5ØØ PRINT"E"
6ØØ END
```



De gebruikte statements en structuren zijn bepaald elementair zodat geen recht wordt gedaan aan de specifieke eigenschappen en voordelen van de betere Basics. Dat heeft echter wel het voordeel dat bijna alle computers de test kunnen doen zodat er inderdaad van vergelijken sprake kan zijn.
Daarom moet bij het "runbaar" maken van de testen ook zo dicht mogelijk bij de oorspronkelijke test worden gebleven.

De Resultaten:

(alle tijden in seconden)

Casio FX702P	18,7	119,7	243,5	238,9	277,1	419,9	643,1	90,5	:2051,4
Sharp PC1500	14,9	28,8	78,0	78,6	92,3	166,7	221,8	50,5	: 731,6
ZX Spectrum	4,9	9,0	21,9	20,7	25,2	68,2	86,7	251,0	: 261,7
TI 99/4A	3,6	9,6	24,6	25,2	26,8	61,0	84,4	38,3	: 273,5
Tandy Color Comp.	2,3	12,3	22,9	24,8	27,9	43,5	63,4	13,5	: 210,6
Dragon 32	1,2	9,1	17,7	19,2	22,2	31,1	44,7	10,8	: 156,0
Commodore 64	1,2	9,3	17,6	19,5	21,0	29,5	47,5	11,3	: 156,9
VIC 20	1,1	8,0	15,2	16,8	18,1	25,4	41,0	9,8	: 135,4
New Brain	2,0	6,0	17,4	17,8	19,9	31,0	49,0	7,4	: 150,5
Fujitsu F8	1,8	6,5	17,2	16,4	17,5	29,1	49,5	6,0	: 144,0
Osborne 1	1,5	4,6	12,1	11,9	12,9	36,1	49,6	6,2	: 134,9
Apple III	1,9	7,2	13,5	14,6	16,0	27,1	42,6	7,6	: 130,5
Sirius 1	1,8	5,3	10,6	10,9	12,6	23,4	35,9	11,3	: 111,8
BBC Micro	0,8	3,1	8,2	8,7	9,1	13,7	21,3	5,3	: 70,2
Acorn Atom	0,6	5,6	10,0	11,3	14,9	20,6	32,6	276,9	: 372,5
	1	3	2	3	4	2	2	15	13

De conclusie is duidelijk:die Floatingpoint van ons is duidelijk voor verbetering vatbaar. Zouden we bijvoorbeeld de Floating point van de Tandy Color Computer hebben (no. 10 bij test 8!) dan was de Atom op de BBC na de snelste. En dat terwijl de BBC op het onderdeel Floating point de snelste is.

J.R.Marks

Vandie dingen ja, vandie dingen....

Nu we het toch over snelheid hebben, nog een klein programma dat de snelheid van het toetsenbord opvoert. Dit is vooral handig in combinatie met de REP-toets. Het programma werkt echter niet voor de cursor-toetsen en de LOCK-toets.

```
 0 REM FAST INPUT
10 DIM LL1;P.$21
20 F.I=1 TO 2;P=#2800
30[PHP;CLD;STX #E4;STY #E5;
40:LL0;BIT #B002;BVC LL1; JSR #FE71;BCC LL0;
50:LL1;LDX @#04;JSR #FB83;JMP #FEA7
60 ]
70 P.$6;N.
80 ?#20A=0;?#20B=#28
90 END
```

Voor snelle jongens mag de 4 op regel 50 worden veranderd in een 1!

FRANS

GOLFVORM

In een aflevering van Acorn Nieuws als deze, waarbij "snelheid" het thema is, mag een vervolg op mijn verhaal "Golfvorm" zeker niet ontbreken.
Dit verhaal concentreerde zich immers op een programma'tje, dat zo snel mogelijk het geluidsbit aan- en uitzette, volgens een bepaald patroon.
Welnu, het gegeven programma kan nòg sneller, aldus Frans van Hoesel. Kijk maar:

```
:LLO  LDA  @BITEEN        /2 µs/  Masker voor "hoog"
:LL1  LDY  @MAX           /2 µs/  MAX is lengte van TABEL
:LL2  STA  #B002          /4 µs/  Zet output "hoog"
      LDX  TABEL,Y        /4 µs/  Haal volgende vertr.tijd op
:LL3  DEX                 /2 µs/  Verlaag teller
      BNE  LL3            /2(3)µs/Nog niet nul? Opnieuw!
      DEY                 /2 µs/  Volgende TABEL-waarde
      STX  #B002          /4 µs/  Zet output "laag"
      LDX  TABEL,Y        /4 µs/  Haal volgende vertr. tijd op
:LL4  DEX                 /2 µs/  Verlaag teller
      BNE  LL4            /2(3)µs/ Nog niet nul? Opnieuw!
      DEY                 /2 µs/  Volgende TABEL-waarde
      BNE  LL2            /2(3)µs/ Vogende periode
      JMP  LL1            /3 µs/  Uitgang; voorlopig: opnieuw!
```

De minimale periodeduur wordt hiermee 31 µs, dus een verbetering met 6 µs (de BNE's heb ik vorige keer verkeerd geteld!). Het verschil is bereikt door het masker BITNUL niet te gebruiken, maar domweg het X-register, dat steeds nul is, op B002 te schrijven. Bovendien wordt de TABEL nu achter-waarts doorlopen, waardoor geen CPY-instructie meer nodig is.
Het voordeel dat we hiermee bereiken, is dat we nu onze golfvorm kunnen opbouwen uit <u>zes</u> trapjes inplaats van <u>vijf</u>, dus een iets minder grove benadering van de gewenste GOLFVORM.

Ton Otten

ERROR in "BENCHMARKING"

In het artikel "BENCHMARKING" van J.R. Marks op pagina 9/10 is helaas door de auteur een nulletje over het hoofd gezien. Hierdoor verandert evenwel de conclusie; nogal drastisch eigenlijk.

Tot zover het <u>slechte</u> nieuws.

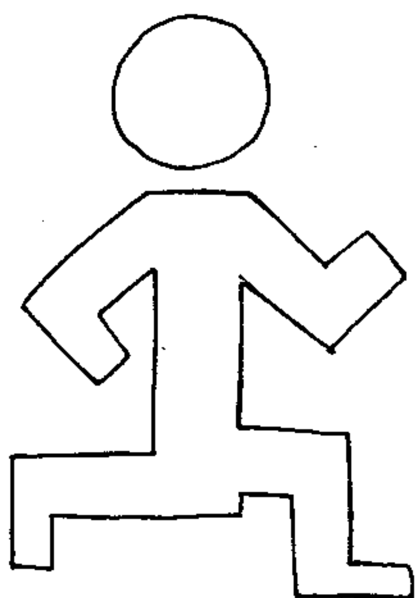
En dan nu het <u>goede</u> nieuws: de opgegeven tijd voor de test nr. 8 (deze test hoofdzakelijk enkele Floating-Point-funkties, HONDERD maal achter elkaar en niet DUIZEND!) op de Acorn Atom is een factor 10 te hoog opgegeven.
Dit betekent dat op het onderdeel Floating Point onze Atom oprukt van de 15^e plaats naar de 11^e en in het Algemeen Klassement van de 13^e plaats <u>naar de 3^e plaats</u>!

Een fraai resultaat, nietwaar?



```
 10 REM REAL-TIME KLOK IN BEELD, ZONDER SOLDEREN !
 20 REM HARDWARE:  6522 (VIA)
 30 Y=176;G=#2800;H=#2800
 40 REM G: RAM-ADRES
 50 REM H: RAM OF (PSEUDO-)ROM ADRES
 60 DIM PP4;P.$21
 70 FOR I=0 TO 4;PPI=0;NEXT I
 80 FOR I=1 TO 2
 90 P=H
100[ LDA@0;LDX@9
110:PP4                             \ INITIALISEER:
120 STA G+#F6,X;DEX;BNE PP4         \ UREN,MINUTEN,SECONDEN = 0
130 LDA@10;STA G+#F9;STA G+#FC      \ VERANDER IRQ-VECTOR
140 LDA @PP0/256;STA #205;LDA @PP0%256;STA #204
150 CLI                             \ MAAK IRQ MOGELIJK
160 LDA@#C0;STA #B80B;STA #B80E     \ INITIALISEER VIA 6522
170 LDA @#F;STA #B804               \ 50000
180 LDA @#C3;STA #B805              \ MICRO-SECONDEN
190 RTS
200:PP0 TXA;PHA;TYA;PHA
210 LDA @#C3;STA #B805              \ RESTART TIMER
220 INC G+#FF;LDA G+#FF;CMP @20;BNE PP2;LDA @0;STA G+#FF
230 INC G+#FE;LDA G+#FE
240 CMP @10                         \ EENHEDEN SECONDEN
250 BNE PP1;LDA @0;STA G+#FE
260 INC G+#FD;LDA G+#FD
270 CMP @6                          \ TIENTALLEN SECONDEN
280 BNE PP1;LDA @0;STA G+#FD
290 INC G+#FB;LDA G+#FB
300 CMP@10                          \ EENHEDEN MINUTEN
310 BNE PP1;LDA@0;STA G+#FB
320 INC G+#FA;LDA G+#FA
330 CMP@6                           \ TIENTALLEN MINUTEN
340 BNE PP1;LDA@0;STA G+#FA
350 INC G+#F8;LDA G+#F8
360 CMP@10                          \ EENHEDEN UREN
370 BNE PP1;LDA@0;STA G+#F8
380 INC G+#F7;LDA G+#F7
390 CMP@10                          \ TIENTALLEN UREN
400 BNE PP1;LDA@0;STA G+#F7
410:PP1 LDX@8
420:PP3                             \ DISPLAY DE TIJD
430 LDA G+#F6,X;CLC;ADC@Y;STA #8017,X;DEX;BNE PP3
440:PP2 PLA;TAY;PLA;TAX;PLA;RTI;]
450 NEXT I
460 P.$6''
470 INPUT"UREN"A;INPUT"MINUTEN"B;INPUT"SECONDEN"C
480 P.$12"                    TIME = "';LINK #
490 REM INITIALISEER DE TIJD!
500 REM INDIEN WEGGELATEN, DAN START MET TIJD = 00:00:00
510 G?#F7=A/10;G?#F8=A%10;G?#FA=B/10
520 G?#FB=B%10;G?#FD=C/10;G?#FE=C%10
530 REM EINDE INITIALISATIE VAN DE TIJD!
540 END
  0 (C) GERARD AKKERMANS
  0      ACC  ROTTERDAM
```

```
>LL.
  10 REM **** JOGGING ****
  20 REM SNELHEIDSTABEL MAKEN             MET ARGUMENTEN:
  30 REM AFSTAND, KORTSTE EN              LANGSTE TIJD, NODIG OM
  40 REM DIE AFSTAND AF TE
  43 REM LEGGEN.
  45 REM ......................
  50 REM OP VERZOEK DE SNELHEID OOK
  60 REM IN MIJLEN PER UUR, OM            MET AMERIKAANSE LIT-
  70 REM TERATUUR TE KUNNEN               VERGELIJKEN.
 100 P.$12'
 110%M=0
 120FIN."AFSTAND"%S,"STARTTIJD"%T,"EINDTIJD"%E,"STAP IN SEC"%D
 130 P.$2
 135 GOS.l;GOS.l
 140P."AFSTAND     ";%X=%S;Y=1;GOS.d;P." KILOMETER, = "
 145 %X=%S/1.609;Y=2;GOS.d;P."   MIJL"'
 150@=0;P."TIJD VAN   ",%T/1," MIN TOT   ",%E/1 " MIN"',"  "'
 160@=8
 170GOS.l
 180P."        MIN     SEC        SNELHEID  SNELHEID"'
 190P."                           KM/H      M/H"'
 200 GOS.l
 210DO
 220   %V=60*%S/(%T+%M/60)
 230%W=%V/1.609
 240FIF%M=0P."  "'
 250   FIF%M=0 P.%T,"",%M,"          ";%X=%V;GOS.d;Y=2
 260 FIF%M=0 P."        ";%X=%W;GOS.d;P.'
 270   FIF%M()0 P."          ",%M,"          ";%X=%V;GOS.d
 280 FIF%M()0 P."       ";%X=%W;GOS.d;P.'
 290   %M=%M+%D
 300   FIF%M=60 %M=0;%T=%T+1
 310UNTIL(%T/1=%E/1)AND(%M/1)0)
 320 GOS.l
 330 P.$3
 340 E.
 350dZ=@;@=0
 360 %X=%X+.005
 370 X=%X/1;P.X,"."
 380 %X=10*(%X-X)
 390 F.W=1TOY
 400    X=%X/1;P.X
 410    %X=10*(%X-X)
 420 N.
 430 @=Z
 440 R.
 450lDO P."-";U.COUNT=40;P.'
 460R.
```



```
>RUN
AFSTAND?12.8
STARTTIJD?56
EINDTIJD?64
STAP IN SEC?20
----------------------------------------
----------------------------------------
AFSTAND    12.8 KILOMETER, = 7.96  MIJL
TIJD VAN   56 MIN TOT   64 MIN

----------------------------------------
        MIN     SEC        SNELHEID  SNELHEID
                           KM/H      M/H
----------------------------------------

        56        0        13.71     8.52
                 20        13.63     8.47
                 40        13.55     8.42

        57        0        13.47     8.37
                 20        13.40     8.33
                 40        13.32     8.28

        58        0        13.24     8.23
                 20        13.17     8.18
                 40        13.09     8.14

        59        0        13.02     8.09
                 20        12.94     8.04
                 40        12.87     8.00

        60        0        12.80     7.96
                 20        12.73     7.91
                 40        12.66     7.87

        61        0        12.59     7.82
                 20        12.52     7.78
                 40        12.45     7.74

        62        0        12.39     7.70
                 20        12.32     7.66
                 40        12.26     7.62

        63        0        12.19     7.58
                 20        12.13     7.54
                 40        12.06     7.50

        64        0        12.00     7.46
----------------------------------------
```

Recurrente betrekkingen voor sinus en cosinus.

Tik de beide volgende, zeer kleine programma'tjes eens in en meet hoeveel tijd ze allebei nodig hebben:

```
10F.I= 1TO1000
20%S=SINI
30N.
40E.
```

```
10%A=0;%S=SIN1;%C=2*COS1
20F.I=2TO1000
30%B=%C*%S-%A
40%A=%S;%S=%B
50N.
60E.
```

Kijk ook naar de waarde die %S heeft gekregen, na afloop van de beide programma's; en vergelijk die met de waarde van de sinus van 1000 radialen.
Het blijkt dat beide programma's hetzelfde doen: het berekenen van de sinuswaarden van 1 t/m 1000 radialen, maar wel met een verschil in tempo!!!!!
Het tweede (snelle) programma maakt gebruik van een zgn. recurrente betrekking: een sinus wordt berekend uit twee voorgaande sinuswaarden. Uiteraard moet er dan aan voorwaarden worden voldaan. In de praktijk blijkt dat dat heel vaak het geval is: juist als er een groot aantal sinusen moeten worden berekend, blijkt dat de waarden waarvoor dat moet regelmatig (hetzelfde bedrag) toe- of afnemen. En dat is juist wat nodig is.
We kunnen dus de recurrente betrekkingen gebruiken voor de berekening van:
SIN(X), SIN(X+H), SIN(X+2*H),......SIN(X+N*H),...... enz. uitgaande van twee zgn. startwaarden.

Hetzelfde geldt ook voor het berekenen van cosinusen.

Wat zijn nu die <u>recurrente betrekkingen</u>?

Die zijn: SIN(X+2*H)=2*COS(H)*SIN(X+H)-SIN(X),
en: COS(X+2*H)=2*COS(H)*COS(X+H)-COS(X).

Voor wie erin geinteresseerd is volgt hier de afleiding.
Er geldt:

SIN(A+B)=SIN(A)*COS(B) + COS(A)*SIN(B)
SIN(A-B)=SIN(A)*COS(B) - COS(A)*SIN(B)

SIN(A+B) + SIN(A-B) =2*COS(B)*SIN(A), of
SIN(A+B) = 2*COS(B)*SIN(A) - SIN(A-B)

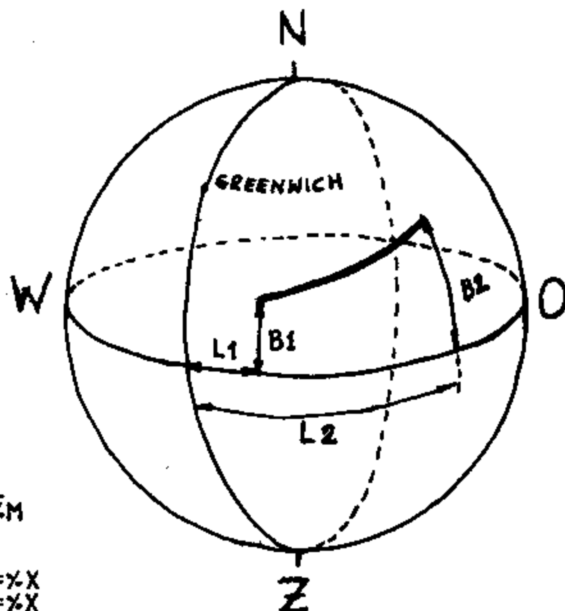
COS(A+B)=COS(A)*COS(B) - SIN(A)*SIN(B)
COS(A-B)=COS(A)*COS(B) + SIN(A)*SIN(B)

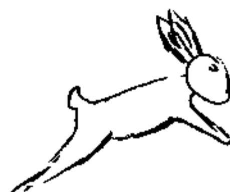
COS(A+B) + COS(A-B) =2*COS(B)*COS(A), of
COS(A+B) = 2*COS(B)*COS(A) -COS(A-B)

Vul nu in: A=X+H, B=H, en je krijgt de genoemde recurrente formules.

Jaap Woldringh.

```
 1 G.90
 5.
 10      A F S T A N D E N O P
 20           A A R D E
 25.
 27      JJH WOLDRINGH
 30      BOTTEMAHEERD 4
 40      9737NE GRONINGEN
 50      TEL.412989
 65.
 90DIMA5;P.$12
 100P."WILT U WERKEN MET HOEKEN"'
 110P."DIE IN GRADEN,MIN EN SEC"'
 120P."ZIJN UITGEDRUKT, OF IN"'
 130P."DECIMALE GRADEN?"'''
 140IN."ANTWOORD GMS OF DG"$A''
 150IF $A="GMS" F=0;G.180
 160IF $A="DG" F=1;G.180
 170P."ANTWOORD ONDUIDELIJK"';G.140
 180P."AFSPRAAK:NOORD, OOST: +"'
 190P."               ZUID,  WEST: -"'''
 200FIN."BREEDTE 1 "%B,"LENGTE  1 "%L
 210FIN.'"BREEDTE 2 "%C,"LENGTE  2 "%M
 220IF F G.500
 225 REM VAN GMS --) DG
 230%X=%B;GOS.g;%B=%X;%X=%L;GOS.g;%L=%X
 240%X=%C;GOS.g;%C=%X;%X=%M;GOS.g;%M=%X
 500 REM VAN GRADEN NAAR RADIALEN
 510%F=PI/180
 520%B=%B*%F;%L=%L*%F;%C=%C*%F;%M=%M*%F
1000 REM DE BEREKENING VAN DE
1005 REM VAN DE AFSTAND
1010%H=ASN(SIN%B*SIN%C+COS%B*COS%C*COS(%L-%M))
1050 REM %H IS DE 'HOOGTE'
1060 REM WAT DAT BETEKENT WORDT
1070 REM IN DE STATEMENTS
1080 REM AAN HET EIND VAN HET
1090 REM PROGRAMMA UITGELEGD.
1200 REM BEREKENING AZIMUTH
1205 REM (DWZ. DE RICHTING)
1210%R=(SIN%C-SIN%H*SIN%B)/(COS%H*COS%B)
1220FIF ABS%R)1 %R=SGN%R
1230%R=ACS%R
1515 REM %H EN %R VAN RADIALEN
1516 REM NAAR GRADEN
1520%H=%H/%F;%R=%R/%F
1522FIF (%M-%L)(0 %R=360-%R
1525 REM AFSTAND (IN GRADEN) IS
1526 REM 90-'HOOGTE'
1530%D=90-%H
1535 REM 360 GRADEN IS 40000 KM
1536 REM OP DE EVENAAR.
1540%D=%D*4E4/360
1545 REM RAD --) DG
1550%B=%B/%F;%L=%L/%F;%C=%C/%F;%M=%M/%F
1560IF F G.2000
1600 REM VAN DG --) GMS
1610%X=%B;GOS.d;%B=%X;%X=%L;GOS.d;%L=%X
1620%X=%C;GOS.d;%C=%X;%X=%M;GOS.d;%M=%X
2000 REM UITVOER
2005Y=4
2006IF F Y=2
2008P.$12
2010P."BREEDTE 1";%X=%B;GOS.r;P.'
2020P."LENGTE  1";%X=%L;GOS.r;P.''
2030P."BREEDTE 2";%X=%C;GOS.r;P.'
2040P."LENGTE  2";%X=%M;GOS.r;P.''
2050FP."AFSTAND IS "%D" KILOMETER"''
2055Z=@;@=0
2060P."AZIMUTH IS "%R+.5" GRADEN."''
2065@=Z
2100END
10000g X=%X/1;%X=100*(%X-X)+SGN%X*9E-6;Y=%X/1;%Z=100*(%X-Y)
10001  %X=X+(Y*60+%Z)/3600;RETURN
20000d X=%X/1;%X=3600*(%X-X);Y=%X/60;Z=%X%60;%X=X+Y/100+Z/10000
20010R.
```





(vervolg op pagina 15)

```
0 REM PREDICTOR
10 @=0;P.$12"   ... PREDICTOR ...""''';?#E1=0
20 P."I'M PROGRAMMED TO PREDICT"
30 P.'"YOUR ACTIONS. EACH TIME"
40 P.'"YOU SHOULD ENTER <T> OR <F>"
50 P.'"AND I WILL UPDATE MY SCORE"
60 P.''"LEVEL (2-8).........?";DO KEY L;L=L-CH"0";U.L>1A.L<9
70 P.$8L
85 PLAY G8
90 ?#23=#00;?#24=#82
100 M=1;F.I=1TOL;M=M*2;N.
110 DIM TTM,FFM
120 F.I=0TOM;TTI=0;FFI=0;N.
130 P.$30''''
140 P."YOUR ENTRY WAS......?        "'
150 P."I WAS PREDICTING....?       "'
160 P."NUMBER OF ENTRIES...0        "'
170 P."CORRECT PREDICTIONS.0 (0.0%) "''''
180 C=1;N=0;R=0;@=0
190 DO ?#E1=#80;DO KEY A;UNTIL A=0
200 IF TTC>FFC THEN B=CH"T"
210 IF TTC<FFC THEN B=CH"F"
215 IF TTC=FFC THEN B=A.R.%2*(CH"T"-CH"F")+CH"F"
230 DO KEY A;UNTIL A=CH"T" OR A=CH"F";Y=4;GOS.a;P.$A
240 GOS.a;P.$B
250 N=N+1
260 GOS.a;P.N" "
270 IF B=A THEN R=R+1
290 GOS.a
300 P.R" ("(R*10000+5)/N/100"."(R*10000+5)/N%100/10"%) ."'''
305 PLAY G8
310 IF A=CH"F" THEN FFC=FFC+1;C=(C*2)&(M-1)
320 IF A=CH"T" THEN TTC=TTC+1;C=(C*2+1)&(M-1)
330 U.0
340a!#DE=Y*32+#148000;Y=Y+1;R.
350.
360 Naar een (oorspronkelijk ?) idee van
370  John Krutch
380
390 Bewerkt door
400  Frans van Hoesel
```

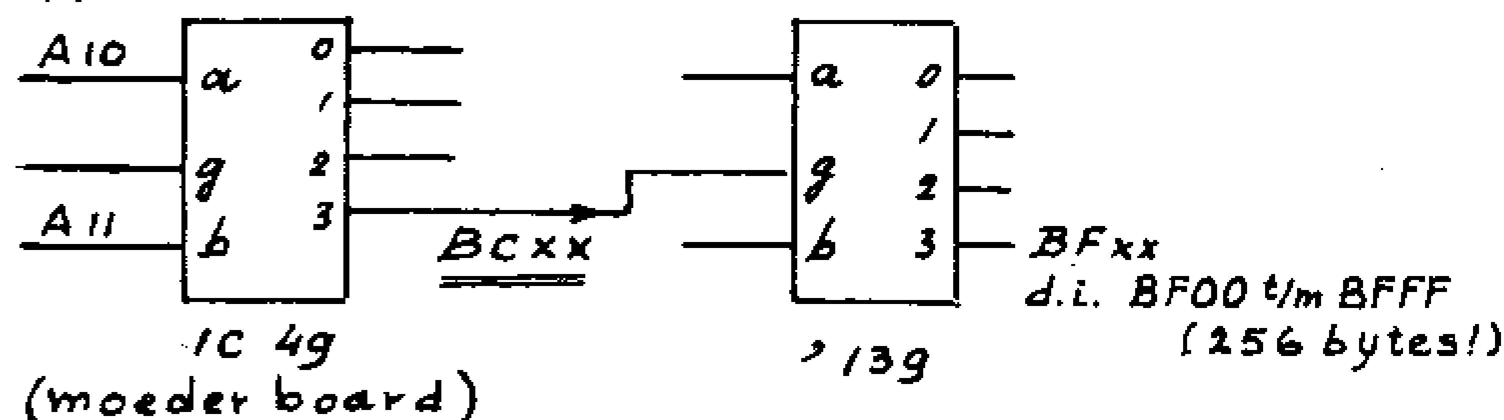
Na de invoer van het level (een geschikt level is 4) steeds na elke piep een 'T' of een 'F' in tikken (niet te vlug). Als dit doet zonder naar het scherm te kijken, dan kan de computer in ongeveer 60% van alle gevallen de invoer correct voorspellen. (dit wordt natuurlijk 50% als U voldoende random kunt tikken). Ook als U naar het scherm kijkt is het lastig om in de buurt van de 50% te komen. Let er echter steeds op dat U niet te snel tikt, dan zou nl. de tijd een random (?) factor kunnen worden, waardoor U gemakkelijker de 50% haalt.

(vervolg "Afstanden op Aarde" van pagina 14)

```
21000rS=SGN%X;%X=ABS%X+.5*10^-Y
21020X=%X/1*S;P.X,"."
21025Z=@;@=0
21030%X=10*(%X-S*X)
21040F.W=1 TO Y
21050  X=%X/1;P.X
21060  %X=10*(%X-X)
21070N.
21080@=Z
21090R.
25000DIT PROGRAMMA KAN OOK WORDEN GEBRUIKT OM DE HOOGTE VAN EEN
25001HEMELLICHAAM(ZON, STER, MAAN, PLANEET), WAARVAN DE DECLI-
25002NATIE EN DE UURHOEK (LOCAAL OF TOV.GREENWICH) BEKEND ZIJN,
25003TE BEREKENEN. DE DECLINATIE IS DAN BREEDTE 2.DE UURHOEK
25004 TEN OPZICHTE VAN GREENWICH(teken!) IS LENGTE 2
25006NIET %D, MAAR %H MOET DAN WORDEN UITGEVOERD.(2050).
25008HET AZIMUTH IS DE RICHTING WAARIN HET HEMELLICHAAM WORDT
25010GEZIEN.
```

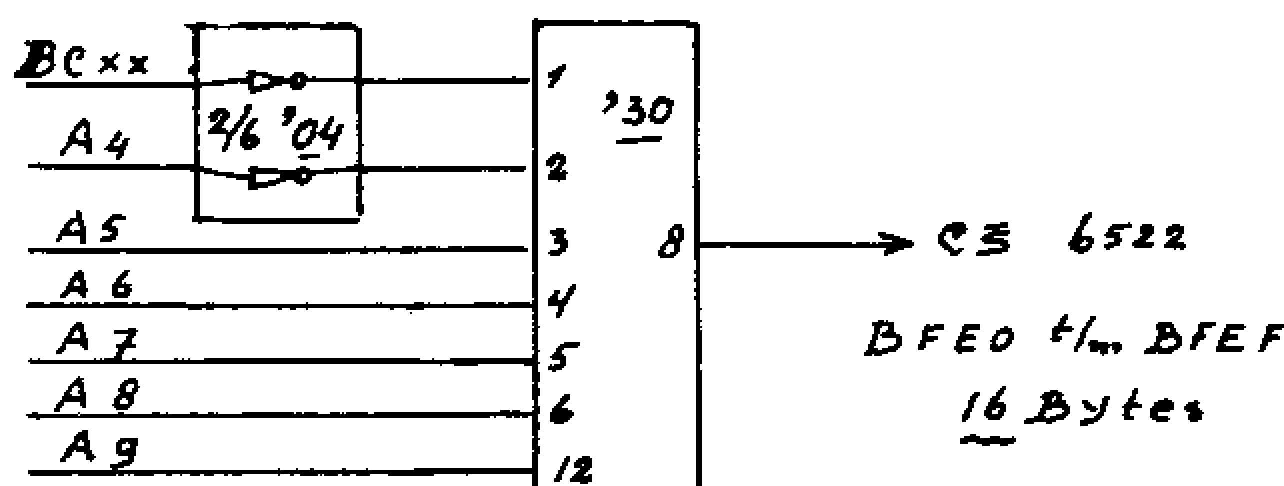
Eén extra 6522 aan de ATOM

In ATOMIX 2, het clubblad van de A.C.Delft, troffen we onderstaand schema aan voor uitbreiding van de ATOM met een extra VIA 6522.



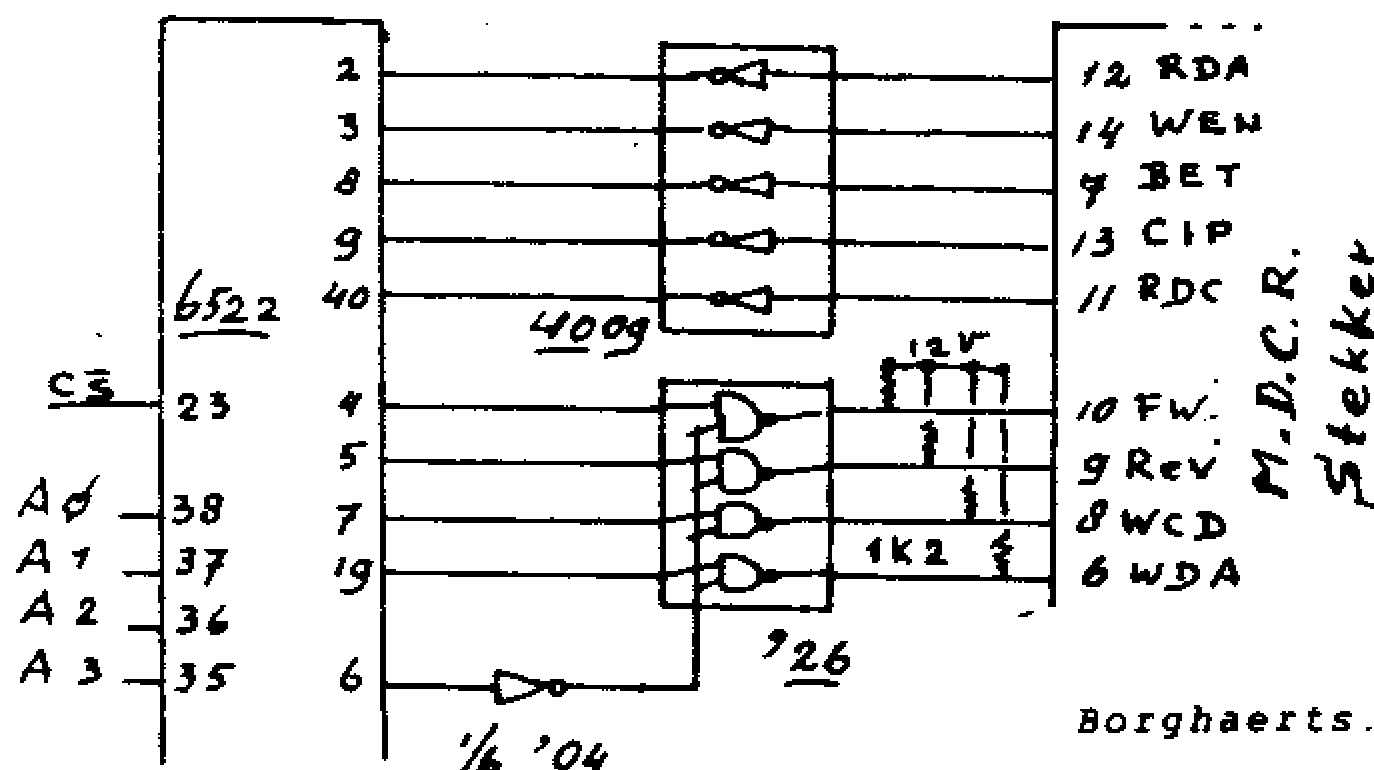
Wij vinden deze oplossing niet fraai. De schakeling 'selecteert' [als je het nog zo noemen mag] 256 bytes, waar een 6522 er maar 16 nodig heeft. Zie MANUAL Blz. 169. Onder de 240 verknalde bytes zit uitgerekend ons SCHAKELBYTE #BFFF. Hoe is't mogelijk.

Dat kan wel beter, zie onderstaand schema [R.Heuvel]



Deze schakeling drijft bij mij een Philips Mini Digitale Recorder aan. Met deze extra VIA [f 18.-] kun je makkelijker de schakelsoft van de PET overnemen zonder dat je last hebt van onze printer op de originele VIA. [Zie program M.v.Peursen in PBE 3-1980].

In hetzelfde PET-bulletin [zie ook aanvulling 9-1982] vindt U onderstaand schema voor de rest van de interface. Dit is een variant van het MANUDAX-schema. Aan een 8255 kan uiteraard ook; zie CHIP Nr 10 [oct.] 1980 met uitleg.



Borghaerts.

WRITE PROTECT OP DE 'REST' VAN DE CMOS-KAART

Wanneer U de CMOS-Geheugenkaart ontvangt dan is het gebied #6800 - #8000 reeds op WRITE-PROTECT te zetten d.m.v. schakelaartje 6. Dit zijn dan de 3 IC's 6,7 en 8. Voor de resterende 5 IC's zijn er dan nog 5 schakelaartjes over. Om te beginnen lieten we U zelf maar uitzoeken hoe dat moest.

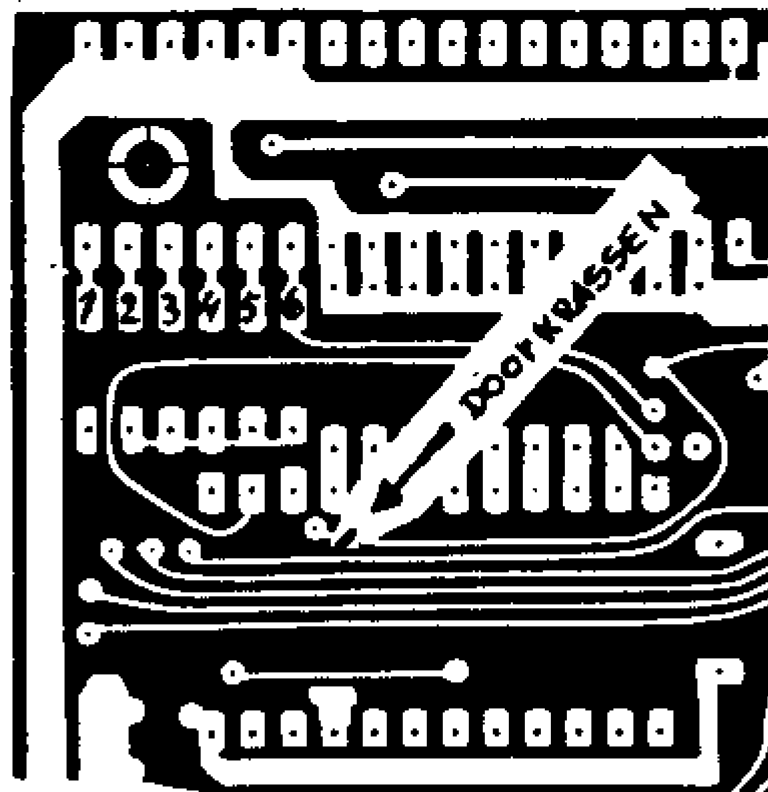
Terecht kennelijk. Met nu een kleine 300 [!] van deze kaarten in de club horen we zelden van problemen. Tot dezer dagen, twee brieven tegelijk met de vraag : " Ik heb nu de 5 lepeltjes aan de IC's 1 t/m 5 aan de schakelaars 1 t/m 5 verbonden [zelfde soldeerpunt waar ook de weerstanden aanzitten], doch dat geeft geen WRITE PROTECT."

Dat klopt en dat zit zo: De IC's 1 t/m 5 zitten, net als alle andere IC's van de ATOM aan één doorlopende $R/\overline{W}$ lijn die wordt gestuurd door de microprocessor. Is de lijn HOOG dan zijn alle IC's intern geschakeld op READ [Uitlezen]. Is de lijn LAAG dan zijn ze intern geschakeld op WRITE [Inschrijven]. LET WEL : In feite is dan echter van alle IC's alleen het éne IC te lezen of in te schrijven,waarvan t e g e l ij k de $\overline{CS}$ lijn LAAG is.

Op WRITE-PROTECT schakelen houdt nu in, dat van een IC, of van een groepje IC's tegelijk [zoals bij IC 6,7 en 8] de $R/\overline{W}$ lijn via een schakelaar loopt.Is die schakelaar gesloten, dan is er niets aan de hand. Bij open schakelaar echter wordt de $R/\overline{W}$ lijn onderbroken en het gedeelte dat naar het IC loopt wordt met een PULL-UP weerstand HOOG getrokken.

Na verbinden van de lepeltjes aan de schakelaars moeten we de IC's derhalve nog losmaken van de oorspronkelijke $R/\overline{W}$ lijn. Dóór de IC voetjes heen ziet U aangegeven, waar de lijn kan worden doorgekrast. Behalve bij IC 1, daar was hiervoor geen plaats.

IC 1 maakt U het beste los aan de soldeerzijde volgens onderstaand tekeningetje.



N.B. Wanneer U het lijntje doorkrast volgens tekening en het naastzittende oogje verbindt met b.v 5 en voor de rest n i e t s doet, dan kunt U met 5 de hele groep IC 1 t/m 5 tegelijk op Write-protect schakelen. Kan ook handig zijn

Borghaerts.

STACK EN QUEUE

Twee veel gebruikte methoden voor tijdelijke opslag van gegevens zijn de stack en de queue (uit te spreken als 'stek' en 'kjoeoe').

Allereerst de queue. Deze is het best te vergelijken met een rij wachtende mensen voor de kassa: wie het eerst in de rij is gaan staan, wordt ook het eerst geholpen. Op z'n Engels heet dat 'first in, first out' (= eersteerin, eerste eruit). Zo'n queue is bijvoorbeeld erg handig in combinatie met een printer: de computer zet alle tekst in de queue, waarbij elke letter achter in de rij aansluit en de printer drukt steeds het eerste karakter in de rij af, waarna het uiteraard uit de rij wordt gehaald. De snelheid waarmee de computer de karakters in de rij zet is niet van belang, zodat de computer gewoon op topsnelheid door kan blijven draaien, terwijl de relatief langzame printer ook z'n werk doet. Natuurlijk moeten we wel in de gaten houden dat de rij niet vol loopt, anders gaan er letters verloren.



Ook het omgekeerde kan het geval zijn: de computer gaat langzamer dan het apparaat. Dit is bijvoorbeeld het geval als we een langzaam programma hebben dat slechts zo af en toe een getalletje van de band (of disk) wil lezen. Het is dan soms handig om in één keer bv. 100 getallen in te lezen en daarmee een queue te vullen. Daarna stopt het cassette-deck (of floppy-disk) en de computer leest één voor één de getallen uit de queue. Als de queue leeg is (de computer heeft alle 100 getallen gehad) dan starten we weer het cassette-deck om de queue weer te vullen. Behalve bij het gebruik van DOS, zit er normaal geen queue in de ATOM.

De stack gedraagt zich net als een stapel borden: het bord dat het laatst op de stapel is gezet, wordt het eerst er weer af gehaald. De stack is handig bij het opbergen van tussenresultaten en bij subroutines. Als voorbeeld nemen we onze Atom; deze heeft verscheidene stack's die ieder een eigen functie

hebben. Een van die stack's is de GOSUB-stack. Hierop wordt het adres gezet van het begin van het volgende statement. Dit wordt gedaan om het RETURN-statement te kunnen uitvoeren; de basic moet dan namelijk weten waar hij weer naar terug moet springen. Voor het opslaan van dat return-adres zijn echter twee geheugenplaatsen nodig, en daarom bestaat de gosub-stack eigenlijk uit twee stack's. De ene stack - waarin het lage byte van het adres wordt opgeslagen - begint op #2CF, en de andere - voor het hoge byte - begint op #2DD.

Bij elke stack hoort een zg. stack-pointer (lett. : stapel-wijzer) die bijhoudt tot hoever de stack gevuld is. Voor de gosub-stack staat die pointer op de geheugenplaats #14. Met behulp van een klein programma en wat denkwerk is een en ander fraai te illustreren:

```
10 P."STACK-POINTER  RETURN-ADRES"'
20 @=10
30 GOS.70;GOS.40;GOS.70;END
40 GOS.70;GOS.50;GOS.70;RETURN
50 GOS.70;GOS.60;GOS.70;RETURN
60 GOS.70;RETURN
70 FOR I=0 TO ?#14-1
80 P.I+1"   "&(I?#2CF+256*(I?#2DD))'
90 NEXT;P.';RETURN
```

De routine op regel 70-90 drukt de gehele gosub-stack af. Het gerommel met plus of min 1 is nodig omdat dit afdrukken in een subroutine gebeurt. Tijdens het runnen van het programma kan men het beste de uitvoer even opschrijven, om die daarna mbv. HEX of XDUMP #2900 te vergelijken met de programma-tekst.

Andere stack's in de Atom zijn oa. :

- De DO-stack: de stack-pointer staat op #13 en de stack zelf bevat het return-adres van de DO-loop op #2B8 en hoger (lage byte) en #2C3 en hoger (hoge byte).
- De FOR-stack: de stack-pointer staat op #15. De stack is 11 bytes breed:

#240 (en hoger)	naam van de for-next variabele
#24B ,,	step-grootte laagste byte
#255 ,,	step-grootte een na laagste byte
#260 ,,	step-grootte een na hoogste byte
#26B ,,	step-grootte hoogste byte

#276 (en hoger)	TO-waarde laagste byte
#281 ,,	TO-waarde een na laagste byte
#28C ,,	TO-waarde een na hoogste byte
#297 ,,	TO-waarde hoogste byte
#2AC ,,	return-adres lage byte
#2AD ,,	return-adres hoge byte

Aangezien de integer getallen in Atom-basic 4 bytes breed zijn, zijn de step-waarde en de to-waarde ook 4 bytes breed.

- De Basic-stack: de stack-pointer staat op #4. De stack is 4 bytes breed:

#16 (en hoger)	laagste byte
#25 ,,	een na laagste byte
#34 ,,	een na hoogste byte
#43 ,,	hoogste byte

AAI! o..

Zoals op onze vorige club-avond is besproken wordt de basic-stack gebruikt voor het bereken van wiskundige expressies, maar ook voor het opslaan van de naam of het adres van een variabele.

- De 6502-stack (ook wel dé stack): de stack-pointer staat niet op een geheugenplaats, maar in de 6502-processor zelf. De stack loopt van #1FF naar beneden tot #100. Dit is zo voor élke 6502-processor. Bij de Atom wordt echter het stuk geheugen van #100 tot #17F als input- en string-buffer gebruikt. Deze stack is zéér belangrijk en men dient dan ook uiterst zorgvuldig met deze geheugenplaatsen om te gaan. De stack kan voor verschillende doeleinden worden gebruikt. De accumulator op de stack zetten gaat met de machine-taal-instructie PHA, er weer vanaf met PLA; Bij een subroutine JSR wordt het return-adres op de stack gezet en met RTS er weer afgehaald; enz.

Sommige programma's hebben een extra stack nodig voor de opslag van gegevens. Een goed voorbeeld daarvan is het programma op pagina 43 van de Atom handleiding.



Frans

O V E R   D E   W E G

Dit programma laat een "rups" over een weg kruipen.
De weg maakt steeds bochten, en U moet de rups in het midden zien te houden.
Met de Z gaat U naar links, en met de M naar rechts.
Als de rups de kant raakt, of als er 50 stippen zijn verschenen, (de rups moet de stippen ophappen) dan is het spel afgelopen.
Het programma maakt in regel 20 gebruik van het trucje wat op bladzijde 33 van Acorn Nieuws nr 1(1983) is beschreven.
Voor geluidseffecten is de Toolkit van Telec noodzakelijk.

```
1Ø P.$12;?≠E1=Ø;?≠8ØØØ=32;N=Ø;O=Ø;@=Ø;U=≠8ØFØ;F.I=1TO16;P.$1Ø;N.
2Ø DIMA14,B14;$A="▮                  ▮";$B="▮     *      ▮";X=9
3Ø CU.X;P.$A'
4Ø C=A.R.%4;IFC=1;X=X+1
5Ø IFC=Ø;X=X-1
6Ø IFC=2;X=X+2
7Ø IFC=3;X=X-2
8Ø IFX<Ø;X=Ø
9Ø IFX>18;X=18
1ØØ CU.X;P.$A'
11Ø KEYZ;IFZ=CH"Z";U=U-1
12Ø IFZ=CH"M";U=U+1
13Ø IFO=5Ø;G.a
14Ø IF?U=≠2A;BE.4Ø,1;N=N+1
15Ø IF?U=≠6A;G.a
16Ø ?U=≠FF
17Ø V=A.R.%1Ø;IFV=2;CU.X;P.$B';IF?U=≠2A;BE.4Ø,1;N=N+1
18Ø IFV=2;?U=≠FF;O=O+1
19Ø G.4Ø
2ØØaF.X=1ØTO255S.1Ø;BE.X,1;N.
21Ø P.$12"Het spel is afgelopen."''"UW SCORE IS "N''
22Ø P."DE MAXIMALE SCORE IS 49"'';?≠E1=≠8Ø;@=8;E.
```

Frank de Groot.

Noot van de redactie:

Voor diegenen die geen Toolkit bezitten kan het commando CURSOR (= CU. in de regels 3Ø, 1ØØ en 17Ø) worden vervangen door: ?≠EØ=X.

Het commando KEY in regel 11Ø, dat overigens ook door de Josbox wordt herkend, kan vervangen worden door:

```
11Ø Z=?≠BOO1; IF Z=191; U=U-1
12Ø IF Z=127; U=U+1
```

De gebruikte toetsen zijn nu: CTRL en SHIFT.
U kunt natuurlijk ook een joystick gebruiken (zie Acorn Nieuws 5 van 1982).

Het vervangen van het BEEP-statement (afgekort: BE.) door een passend geluidssignaal (wat dacht u van een politie-sirene?) laat ik aan uw eigen fantasie en vaardigheid over.

WISKUNDE

Van de hand van de heer J. van der Graaf ontvingen wij een bundel wiskundig georiënteerde programma's, waarvan we hier een eerste gedeelte opnemen. Het tweede deel, dat wat specialistischer van aard is, volgt in het eerstvolgende nummer van Acorn Nieuws.

De eerste twee programma's, genaamd "STRINGMANIPULATIE", demonstreren het effect van enkele string-bewerkingen.
Deze programma's spreken voor zichzelf en kunnen gebruikt worden als uitgangspunt voor verder string-knutselwerk.

Het derde programma CONVERSIE voert een zg. ster-driehoektransformatie uit, iets wat te pas komt bij de elektrotechniek van driefase-schakelingen. Verdere toelichting bij het programma!

Tenslotte een nieuwe en een oude benaderingsmethode voor nulpunten van algebraïsche functies. De nieuwe methode is die van de Regula Falsi, de oude methode is de reeds eerder beschreven Interval-halveringsmethode, ditmaal van de hand van Jaap Woldringh. De eerste publicatie over dit onderwerp verscheen in Acorn Nieuws van april, van Hans van der Lint, tegelijk met die van Jaap in Acorn Nieuws Noord. Een helder betoog, dat ik u niet wil onthouden.

```
 10 REM STRING MANIPULATIE
 20 REM J V/D GRAAF
 30 REM MAASLAND
 40 DIM A(63)
 50 IN."VOER UW TEKST IN"'$A'
 60 IN."VERTRAGING"'D,'
 70 CLEAR Ø;?#E1=Ø
 80 FOR B=Ø TO LEN(A)-1
 90 PRINT"             "$A?(B)'
100 FOR C=Ø TO D
110 WAIT;NEXT C;NEXT B;GOTO8Ø
```

programma lengte: 242 bytes

invoer voorbeeld:
RUN
VOER UW TEKST IN
?ACORN ATOM-

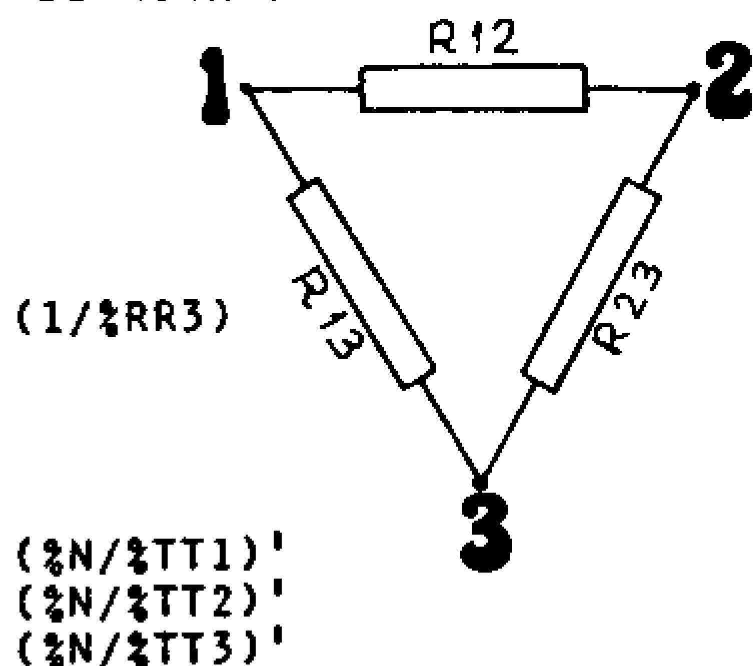
VERTRAGING
?15

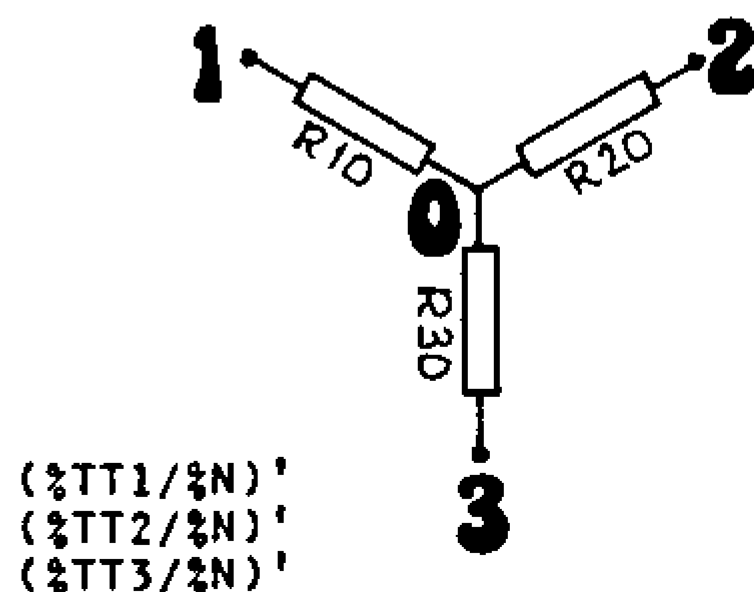
```
 10 REM STRING MANIPULATIE 2
 20 REM J V/D GRAAF
 30 REM MAASLAND
 40 DIM A(63),X(63)
 50 IN."VOER UW TEKST IN"'$A'
 60 IN."VERTRAGING"'D,'
 70 CLEAR Ø;?#E1=Ø;PRINT'
 80 FOR B=Ø TO LEN(A)
 90 $X=$A;$X=$X+B;$X+1=""
 95 PRINT $X
100 FOR C=Ø TO D
110 WAIT;NEXT C;NEXT B;GOTO7Ø
```

programma lengte: 264 bytes

probeer ook eens met toevoeging van de volgende regel:
94 CLEAR Ø

```
)LIST
   1Ø REM CONVERSIE
   2Ø REM J V/D GRAAF
   3Ø REM MAASLAND
   4Ø DIMS(2);FDIM%RR(3),%TT(3)
   5Ø PRINT"STER NAAR DRIEHOEK""(SD)OF"'"DRIEHOEK NAAR "
   6Ø INPUT"STER(DS)"'$S'
   7Ø PRINT"DE INDICES STELLEN ""DE HOEKPUNTENVOOR."
   8Ø PRINT"MIDDELPUNT IS Ø"''
   9Ø IF $S="DS" THEN GOTO 23Ø
  1ØØ PRINT"STER INVOER(OHM)"'
  11Ø FINPUT"WEERSTAND R1Ø"%RR1
  12Ø FINPUT"WEERSTAND R2Ø"%RR2
  13Ø FINPUT"WEERSTAND R3Ø"%RR3
  14Ø %N=(1/%RR1)+(1/%RR2)+(1/%RR3)
  15Ø %TT1=(1/%RR1)*(1/%RR2)
  16Ø %TT2=(1/%RR2)*(1/%RR3)
  17Ø %TT3=(1/%RR1)*(1/%RR3)
  18Ø P."DRIEHOEK UITVOER(OHM)"'
  19Ø FPRINT"WEERSTAND R12="(%N/%TT1)'
  2ØØ FPRINT"WEERSTAND R23="(%N/%TT2)'
  21Ø FPRINT"WEERSTAND R13="(%N/%TT3)'
  22Ø END
  23Ø P."DRIEHOEK INVOER(OHM)"'
  24Ø FINPUT"WEERSTAND R12"%RR1
  25Ø FINPUT"WEERSTAND R13"%RR2
  26Ø FINPUT"WEERSTAND R23"%RR3
  27Ø %N=%RR1+%RR2+%RR3
  28Ø %TT1=%RR1*%RR2
  29Ø %TT2=%RR1*%RR3
  3ØØ %TT3=%RR2*%RR3
  31Ø PRINT"STER UITVOER(OHM)"'
  32Ø FPRINT"WEERSTAND R1Ø="(%TT1/%N)'
  33Ø FPRINT"WEERSTAND R2Ø="(%TT2/%N)'
  34Ø FPRINT"WEERSTAND R3Ø="(%TT3/%N)'
  35Ø END
)
```





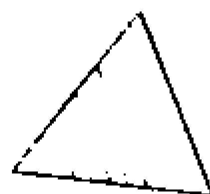
programma lengte:1062 bytes

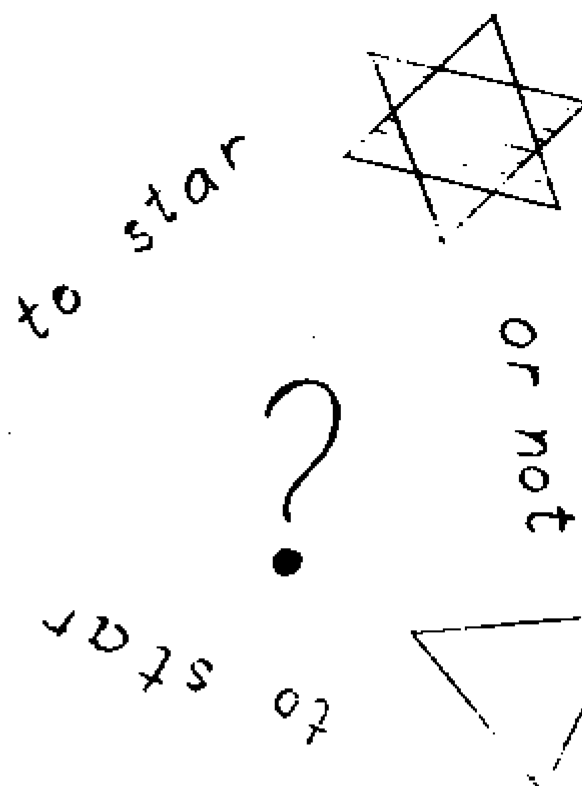
invoer voorbeeld:

```
RUN
STER NAAR DRIEHOEK(SD)OF
DRIEHOEK NAAR STER(DS)
?DS

DE INDICES STELLEN DE HOEKPUNTEN
VOOR.MIDDELPUNT IS 0

DRIEHOEK INVOER(OHM)
WEERSTAND R12?1000
WEERSTAND R13?1000
WEERSTAND R23?500
STER UITVOER(OHM)
WEERSTAND R10=400.00000
WEERSTAND R20=200.00000
WEERSTAND R30=200.00000
```





Nulpunts bepaling met de Regula Falsi

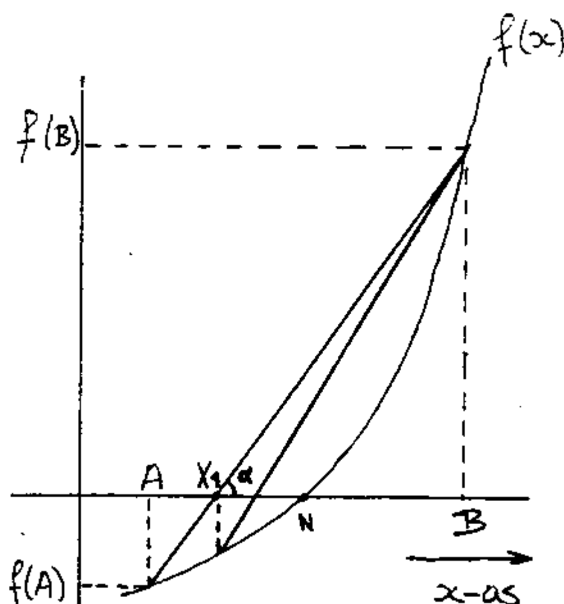
We wensen het nulpunt te vinden dat we in de schets hebben aangegeven met N.We weten dat N tussen A en B ligt.
A is de ondergrens,B is de bovengrens.

De werkwijze.

We trekken de rechte lijn van het punt (A,f(A)) naar het punt (B,f(B)).Het snijpunt van deze getrokken rechte met de x-as noemen we X1.

A en B liggen ieder aan een andere kant van N.Er moet dus noodzakelijkerwijs gelden dat $f(A) \cdot f(B) \leqslant \emptyset$.

Nu we X1 hebben gevonden willen we weten ofdat N tussen A en X1 of tussen X1 en B ligt.
Als N tussen A en X1 ligt,dan moet gelden dat $f(A) \cdot f(X1) \leqslant \emptyset$.
Als N tussen X1 en B ligt,dan moet gelden dat $f(X1) \cdot f(B) \leqslant \emptyset$.
Door te testen kunnen we er achter komen waartussen N nu ligt.



In het getekende geval ligt N tussen X1 en B.We hebben dus het interval waarin N ligt kunnen verkleinen.
Deze nulpunts bepaling is een benadering,we zullen dus voor we beginnen een zekere marge aan moeten nemen waarbinnen de benadering van het nulpunt moet komen te liggen.
We zouden bv kunnen wensen dat het benaderde nulpunt op 1/1000 nauwkeurig is.Deze nauwkeurigheid noemen we epsilon(eps).

We testen nu ofdat X1 al dicht genoeg bij het nulpunt ligt,dit gaat alsvolgt: $f(X1+eps) \cdot f(X1-eps) \leqslant \emptyset$ .Als dit inderdaad zo is dan is X1 de benadering voor het nulpunt N en zijn we klaar.
Als deze test niet positief is dat gaan we verder met de benadering.
Wat gaan we nu doen?
Dat hangt af in welk interval N nu ligt.
Als geldt $A \leqslant N \leqslant X1$ dan is de boven-grens dus veranderd, B moet dus gelijk gemaakt worden aan de waarde van X1.
Als geldt $X1 \leqslant N \leqslant B$ dan is de onder-grens dus veranderd, A moet dus gelijk gemaakt worden aan de waarde van X1.

In ons geval moet de onder grens aangepast worden,dus A=X1.
We kunnen nu het verhaal weer oppakken bij het begin.
We trekken nu weer een lijn van de nieuwe f(A) naar f(B) ,dit is in de tekening reeds gedaan.Het snijpunt van deze lijn met de x-as wordt weer X1 genoemd,enz. .

We hebben nu het principe besproken en gaan verder met de afleiding van een algemene formule.

Uit de tekening zien we dat $\tan\alpha = \frac{f(B)-f(A)}{B-A}$
voor de koorde geldt de vergelijking $y=f(A)+\frac{f(B)-f(A)}{B-A}(x-A)$
Als de koorde de x-as snijdt geldt er $y=\emptyset$

$$\text{oftewel } \emptyset=f(A)+\frac{f(B)-f(A)}{B-A}(x-A)$$
$$-f(A)=\frac{f(B)-f(A)}{B-A}(x-A)$$
$$\frac{-f(A)\cdot(B-A)}{f(B)-f(A)}=(x-A)$$
$$x=A-\frac{f(A)\cdot(B-A)}{f(B)-f(A)}$$
$$x=\frac{A\cdot(f(B)-f(A))-f(A)\cdot(B-A)}{f(B)-f(A)}$$
$$x=\frac{A\cdot f(B)-A\cdot f(A)-B\cdot f(A)+A\cdot f(A)}{f(B)-f(A)}$$
$$x=\frac{A\cdot f(B)-B\cdot f(A)}{f(B)-f(A)}$$

We hebben nu de formule afgeleid die het nulpunt van de koorde X1 berekent.
We zijn nu in staat om met deze gegevens de nulpuntsbepaling in een programma te zetten.

Toelichting op de invoer van het programma REGFAL:
Boven en onder grens zullen nu wel duidelijk zijn,
de toegestane afwijking is de nauwkeurigheid epsilon die we wensen,het aantal slagen komt overeen met het aantal malen dat de koorde getrokken dient te worden.
Als we van de functie $g(y)=z^2-5$ de nulpunten willen bepalen, dan vullen we op regel 320 van het programma de volgende formule in %F=%X*%X-5 .Nadat we de grenzen hebben vastgesteld kunnen we de nulpunten bepalen.

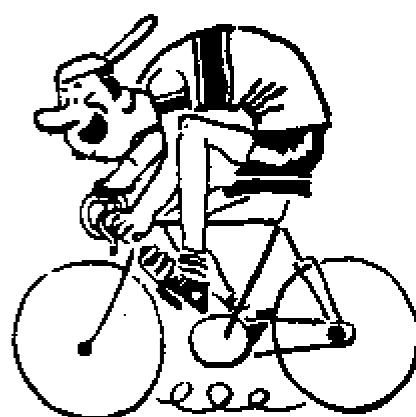
```
10 REM REGULA FALSI
20 REM J V/D GRAAF
30 REM HELSLAND
40 FIN."BOVENGRENS"%B
      ,"ONDERGRENS"%A
50 FIN."TOEGESTANE"
      " AFWIJKING"%E
60 IN."MAX. AANTAL SLAGEN"A
70 B=Ø;∂=1;P.'"ANTWOORD:"'
80 %X=%A;GOSUBf;%C=%F
90 %X=%B;GOSUBf;%D=%F
100 FIF %C*%D<=Ø;GOTO 130
110 P."GEEN OF EEN EVEN AANT"
    "AL NUL-"'"PUNTEN OF EEN"
120 P."RAAKPUNT TUSSEN"'"DE "
    "OPGEGEVEN GRENZEN."';END
130 FIF %C=Ø;%Y=%A;GOSUBb
140 FIF %D=Ø;%Y=%B;GOSUBb
150 DO B=B+1
160 %Y=(%A*%D-%B*%C)/(%D-%C)
170 %X=%Y+%E;GOSUBf;%G=%F
180 %X=%Y-%E;GOSUBf;%H=%F
190 FIF %G*%H<=Ø:GOTOa
200 %X=%Y;GOSUBf;%H=%F
210 FIF %C*%H<Ø;%B=%Y;%X=%B;
    GOSUBf;%D=%F
220 FIF %C*%H>Ø;%A=%Y;%X=%A;
    GOSUBf;%C=%F
230 UNTIL B=A
240 P."DE TOEGESTANE AFWIJ"
    "KING IS NIET BEREIKT."'
250 GOSUBb
260aPRINT"NA "B" SLAGEN IS DE"
    "TOEGESTANE"'
270 P."AFWIJKING BEREIKT."'
280b%X=%Y;GOSUBf
290 FP."DE BENADERING IS "%Y
    '"DE FUNCTIEWAARDE "%F'
300 END
310 REM:UW VERGELIJKING %F ALS
    FUNCTIE VAN %X OP LYN 320
320f%F=EXP(%X)-7
330 RETURN
```

programma lengte:1028 bytes

invoer voorbeeld:
We wensen het nulpunt van de functie %F=EXP(%X)-7. Hiertoe vullen we deze functie op lyn 320 in.We weten dat het nulpunt tussen x=3 en x=1 ligt.

```
RUN
BOVENGRENS?3
ONDERGRENS?1
TOEGESTANE AFWIJKING?1E-6
MAX. AANTAL SLAGEN?20

ANTWOORD:
NA 17 SLAGEN IS DE TOEGESTANE
AFWIJKING BEREIKT.
DE BENADERING IS 1.94590930
DE FUNCTIEWAARDE -5.93438744E-6
```



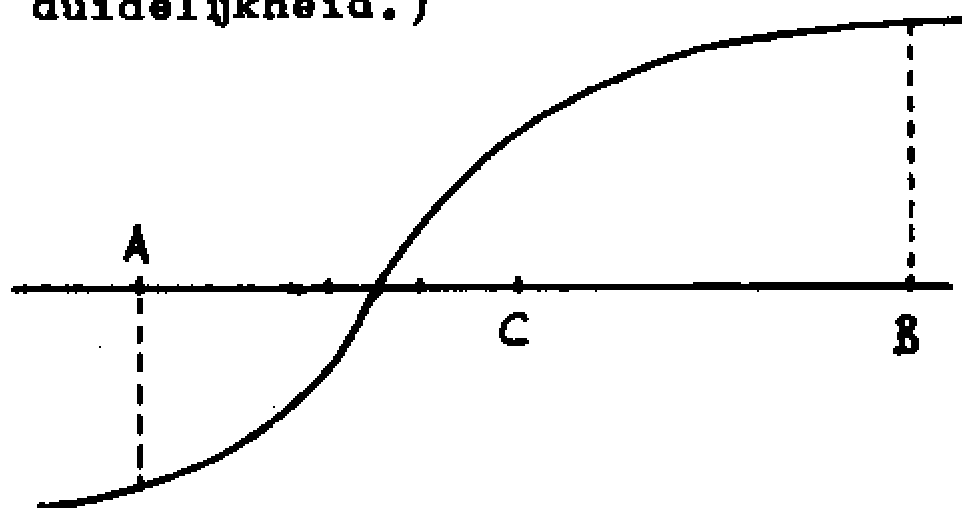
BEREKENING VAN NULPUNTEN MET BEHULP VAN DE INTERVAL-HALVERINGSMETHODE

In de Acorn Nieuws Noord van December 1982 beschreef Henk Boxma de Newton Raphson methode voor het berekenen van de nulpunten van een gegeven functie. In het Januari nummer heb ik daarop nog iets geschreven over het met behulp van de computer berekenen van de afgeleide functiewaarden die daarvoor nodig zijn.
Hoewel deze methode erg snel tot resultaten kan leiden, is het niet zo dat we nu voorgoed uit de problemen zijn als we van functies nulpunten willen bepalen. Dat is niet het geval en Henk heeft in zijn verhaaltje al het een en ander over verteld.

Eén heel ander soort methode, die veel gemakkelijker te begrijpen is, maar niet zo snel tot nauwkeurige resultaten leidt als de Newton Raphson methode, is de zg. Interval-halverings methode. Ook hier hebben we een schatting nodig van het gezochte nulpunt, waarna we links en rechts ervan een waarde kiezen, die dus de grenzen zijn van een interval, dat het gezochte nulpunt bevat.
We gaan nu een leuk spelletje spelen: het interval gaan we in twee gelijke stukken verdelen. Daarna gaan we na of het linkse interval het gezochte nulpunt bevat, door het product van de functiewaarden van de eindpunten te bepalen: is dat product <0 dan bevat dat interval het nulpunt, anders het andere interval (als het product =0 is, is het laatst bepaalde eindpunt van het interval het gezochte nulpunt!!!!).
Het interval dat het nulpunt bevat, wordt ons nieuwe interval.
Dit proces kan je net zo lang herhalen, tot het gezochte nulpunt voldoende dicht benaderd is (dwz.het omvattende interval voldoende klein is geworden).

Stel, bijv. dat het gezochte nulpunt tussen 3 en 4 ligt en de functie van X in het programma in de subroutine f staat die de functie waarde Y retourneert. We kunnen dan het volgende programma schrijven: (zonder %-tekens vanwege de duidelijkheid.)

```
100 A=3;B=4
110 C=(A+B)/2
120 X=A;GOS.f;F=Y
130 X=C;GOS.f;F=F*Y
140 IF F<0B=C;G.170
150 IF F=0P."NULPUNT=",C;E.
160 A=C
170 IF (B-A)>1.E-5G. 110
180 P."NULPUNT=",(A+B)/2
190 E.
```



Ook deze methode is (natuurlijk) behept met slechte eigenschappen. Allereerst is hij tamelijk langzaam. Verder moet de functie zich binnen het interval waar we van uitgaan netjes gedragen en er niet meerdere nulpunten bevatten, of discontinu zijn bijvoorbeeld.
Ook als het nulpunt een raakpunt aan de X=as is gaat het mis. In feite hebben we dan trouwens te maken met twee samenvallende nulpunten. (Ga dat maar eens na!)

Ik hoop dat dit enkele leden kan helpen, die de Newton Raphson methode wat moeilijk vonden. Groeten van

Jaap Woldringh.

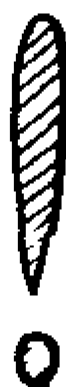
Geachte redactie , mede-hobbyisten

deze maand wegens tijdgebrek slechts een kleine bijdrage .

Allereerst twee opmerkingen

1 : ik vind het zonder meer storend dat ik in mijn lijfblad her en der verspreid aantref ")L." met daarop volgend een of ander programma(atje) zonder enig commentaar of toelichting . Weet , heren programmeurs , dat de waarde van een programma mede afhangt van zijn dokumentatie !!

2 : gezien het aantal leden dat blijkbaar in staat is kant en klare kopij in te zenden lijkt het mij handig als u (redactie) aangaf in welk formaat u het aangeleverde materiaal bij voorkeur zou ontvangen.



Dit is mij, uw redacteur, uit het hart gegrepen!

ad.1 Helemaal mee eens.

ad.2 Zie A.N.1(1983), pag.53:"Advies aan Auteurs";(257x170mm^2).

Okee , de programmaatjes

als eerste een klein en snel (34 milliseconden) stukje machinetaal om uw grafische scherm 'clear4' te inverteren . Zo'n programma hebben we natuurlijk allemaal , maar dit is het snelste wat ik bij kompakte afmetingen wist te produceren .

als tweede een routine om uw grafische scherm 'clear4' te spiegelen t.o.v. de vertikale middellijn . Hoe een en ander werkt ? Een regel van 32 bytes wordt in twee helften verdeeld . Er wordt een 16*8+1=129 (16 bytes + carry) bits rotatie naar links uitgevoerd op de linker helft van de regel ; het uitschuivende bit (carry) wordt aan de linkerkant van de rechterhelft ingeroteerd , waarbij het bij deze rechtse rotatie uitschuivende bit aan de rechterkant van de linker helft wordt ingeroteerd . Dit gebeurt 128 maal , voordat een regel gespiegeld is . Byte #93 doet dienst als tijdelijke drager van de carry omdat de carry verloren gaat bij de CPY-instruktie .
Dit hele verhaal gebeurt dus 192 keer voordat het hele scherm gespiegeld is .

U moet niet verwachten dat het programma in enkele milliseconden zijn werk gedaan heeft ; er gaan een kleine 25 seconden mee heen . Als u dat lang vindt dan moet u maar eens aan het tellen slaan : regel 70 t/m 100 en regel 110 t/m 140 worden elk zo'n kleine 400000 keer doorlopen , wat neerkomt op ruim 2,3 miljoen rotaties/shifts ad 5 cycles , nog afgezien van de indirecte geheugenbewerkingen .

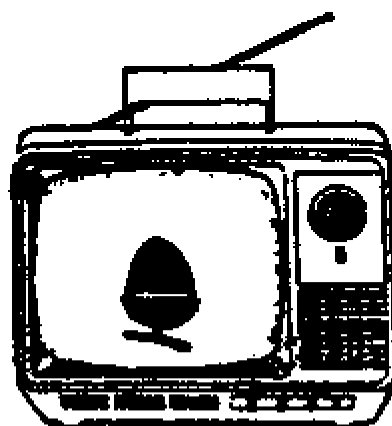
Tot slot nog een optisch grapje (eerlijkheidshalve : niet zelf verzonnen . Oorspronkelijk afgedrukt in de Telegraaf . Wel heb ik het raadsel zelf gekraakt) .
Neem een spiegel van ca. 20 cm. hoog , minstens 10 cm. breed en plaats deze midden tussen de beide vierkanten .
Plaats uw neus tegen de bovenkant van de spiegel en kijk met het oog aan de blinde kant van de spiegel rechtstreeks naar het vierkant en met het andere oog via de spiegel naar het andere vierkant .
Probeer door te bewegen met het hoofd de beide beelden in elkaar te laten overvloeien en 'geniet' van de illusie .
Voor de volgende A.N. zal ik het programma insturen waarmee m.b.v. een grafische printer deze en andere illusies te produceren zijn .

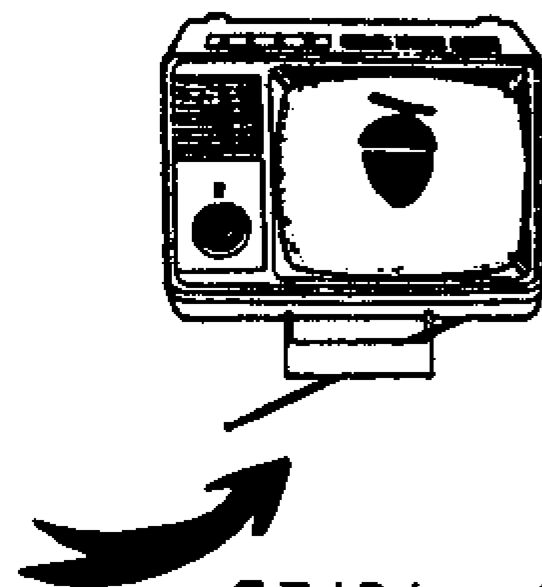
Veel plezier ermee !

Rob van Dort

inverteer scherm routine : 34 millisec.

```
  0   REM inverteer scherm routine : 34 millisec.
 10   DIM LL(1)
 20   P=#7100
 30   [
 40   :LL0 LDX@0
 50        LDA@#80;STA LL0+14;STA LL0+19
 60        LDY@24
 70   :LL1 LDA#8000,X;EOR@#FF
 80        STA#8000,X
 90        INX;BNE LL1
100        INC LL0+14;INC LL0+19
110        DEY;BNE LL1
120   RTS
130   ]
140   END
```





ZZIP!...?!

PROGRAMMALENGTE 265 BYTES

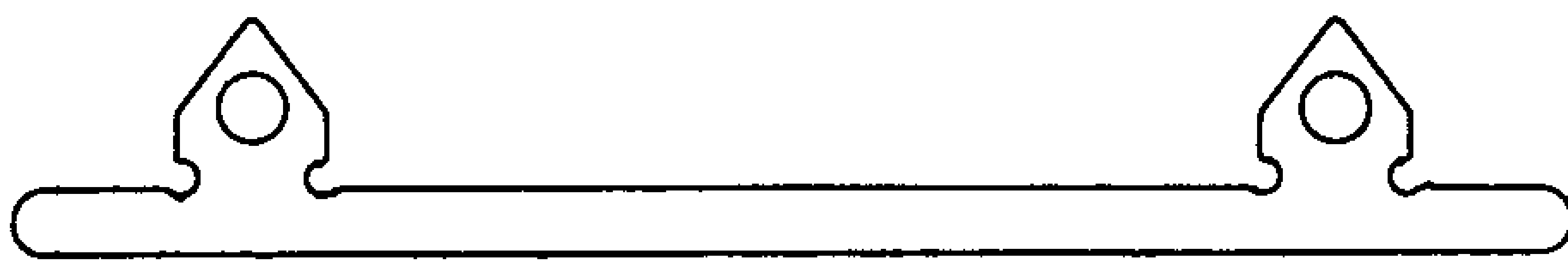
*** spiegel scherm routine ***

```
  0   REM spiegel scherm routine
 10   P=#7000
 20   DIMNN10
 30   [
 40   :NN0 LDA@0;STA#90;LDA@#80;STA#91
 50   :NN1 LDX@0
 60   :NN2 LDY@16
 70   :NN3 DEY
 80        LSR#93;LDA(#90),Y;ROLA;STA(#90),Y;ROL#93
 90        CPY@0;BNENN3
100        LDY@15
110   :NN4 INY
120        ROR#93;LDA(#90),Y;RORA;STA(#90),Y;ROL#93
130        CPY@31;BNENN4
140        INX;CPX@128;BNENN2;LDY@16
150   :NN5 DEY;LSR#93
160        LDA(#90),Y;ROLA;STA(#90),Y;ROL#93
170        CPY@0;BNENN5
180        LDA#93;LDY@15;ORA(#90),Y;STA(#90),Y;LDA@0;STA#93
190        LDA#90;CLC;ADC@32;STA#90;LDA@0;ADC#91;STA#91
200        LDA#91,CMP@#98;BNENN1
210   RTS
220   ]
230   E.
```

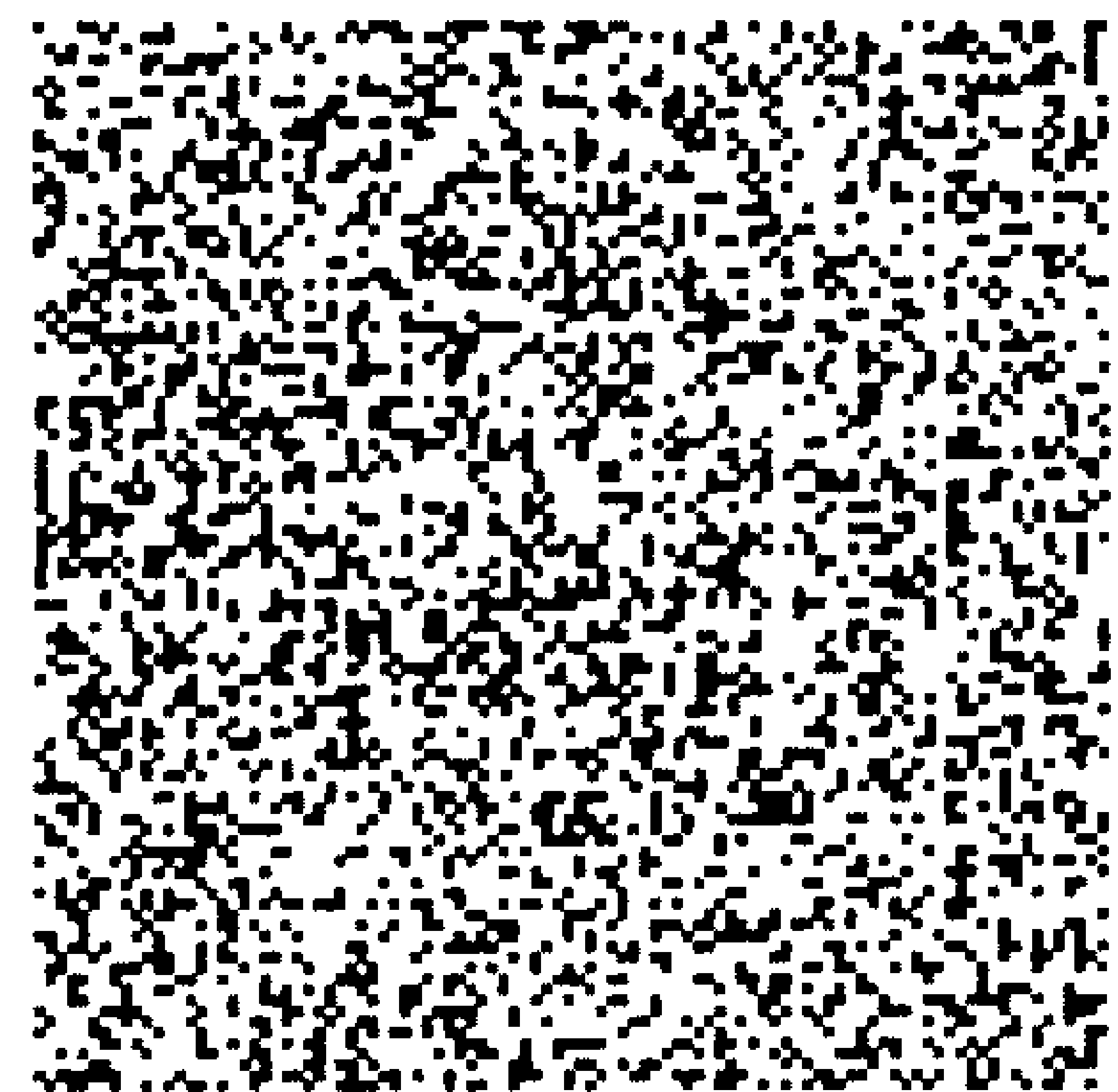


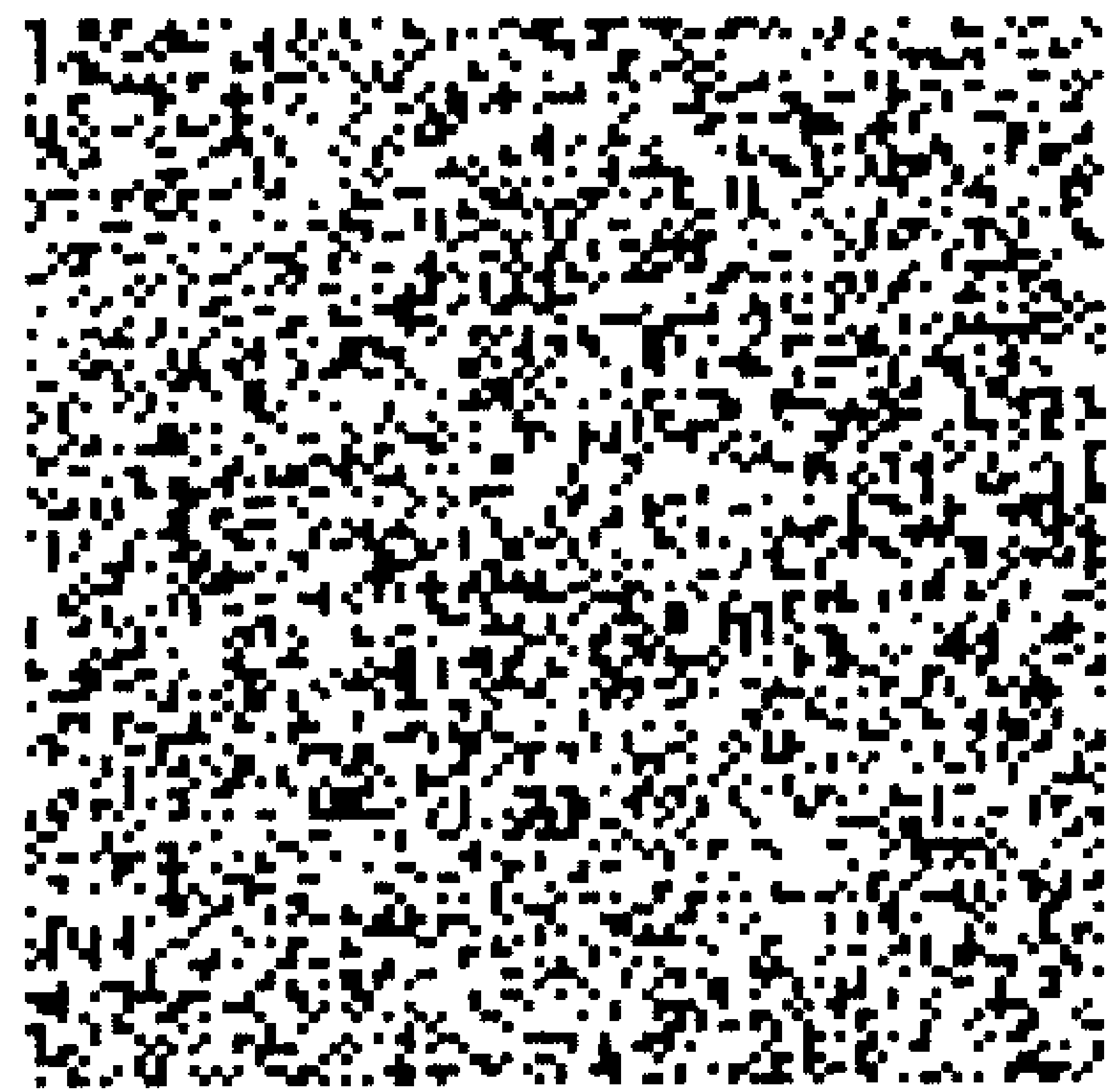
Spiegeltje,
spiegeltje..

PROGRAMMALENGTE 554 BYTES



Als u dit metalen voorwerpje van binnenuit door de rug van uw Acorn Nieuws naar buiten drukt, vanuit de middenpagina, dan kunt u het boekje in een normale ordner opbergen, ZONDER GAATJES! Bij de kantoorboekhandel.





Een fundamenteel probleem !?

Bij het bedenken en programmeren van nieuwe commando's is Gerard Akkermans op een interessant probleem gestoten dat ons aller aandacht verdient, namelijk:

Waarom moet het programma, dat bij een bepaald statement behoort, soms worden afgesloten met een RTS en in andere gevallen met een JMP? Aan het programma zelf is dat niet te zien!

Ter verdere inleiding laten we Gerard zelf even aan het woord:

> Wat betreft de *-commando's:
> - De firma "WILLOW SOFTWARE" brengt een Toolbox op de markt, die ook met *-commando's werkt.
>
>   Na enig snuffelen kwam ik tot de conclusie:
>   - Sommige commando's <u>moeten</u> met <u>RTS</u> worden afgesloten om geen foutmelding te krijgen.
>   - Sommige commando's <u>moeten</u> met een <u>JMP</u> worden afgesloten om geen foutmelding te krijgen.
>
>   Vraag: Wat bepaalt of een commando met een JMP of met een RTS moet worden afgesloten??
>
>   Ikzelf weet het antwoord <u>nog</u> niet. Misschien iemand anders wel!? Het lijkt mij verstandig om in het Acorn Nieuws hieraan even aandacht te besteden.
>
>   (Voorbeelden: *ZERO verlangt een RTS.
*DEL verlangt een JMP.
Tot deze conclusie kwam ik door het gewoon te proberen!)
>
> Met vriendelijke Groet,
>
> Gerard Akkermans
>
> pS. Zero stelt alle variabelen op 0 en Del verwijdert regels.

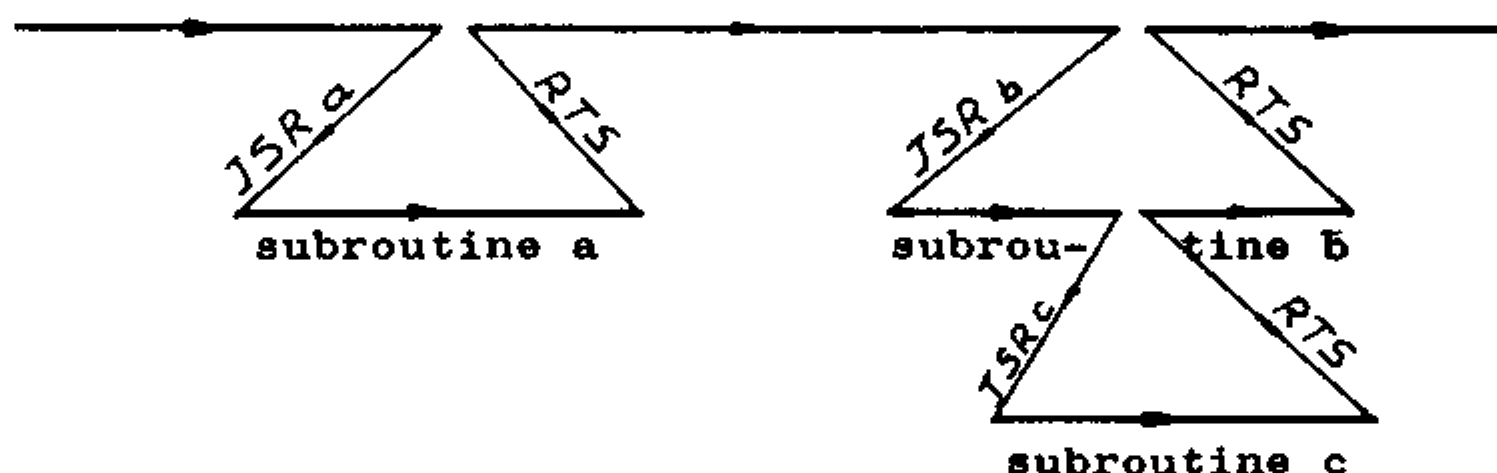
Ik neem aan dat het probleem nu duidelijk is.
En dan nu de oplossing, die m.i. van fundamenteel belang is bij het correct invoeren van nieuwe statements, iets waarmee velen van ons op dit moment druk doende zijn (dacht ik).

Bezien we, om te beginnen, de volgende omschrijving van een subroutine: Voor de uitvoering van een subroutine worden de huidige activiteiten even onderbroken. Vervolgens wordt de subroutine uitgevoerd, waarna de huidige activiteiten weer worden hervat.

De clou zit 'm in dat onderbreken en hervatten: na de subroutine gaat het programma altijd weer door waar het 'gebleven' was.
In BASIC betekent dit met het eerstvolgende regelnummer na de betreffende GOSUB, in ASSEMBLER met de instructie, volgend op de JSR.

Hoe 'weet' die processor nu waar hij gebleven was?
Antwoord: het terugkeeradres en eventueel de inhoud van de accumulator worden bewaard op de 'stack' tijdens de uitvoering van de subroutine. (Voor meer informatie over de 'stack', zie de assemblercursus, pag. 37 en/of pag. 20). Vlak vóór en vlak ná de uitvoering van de subroutine staat de 'stack' dus precies even hoog.

Het voorgaande kunnen we in een plaatje weergeven, als we het doorlopen van het (hoofd-)programma voorstellen door het lopen langs een lijn:



Op deze wijze wordt nu ook duidelijk, waarom een RTS aan het eind van een subroutine moet staan, namelijk om te zorgen dat de processor de draad op het juiste punt weer oppakt. Zou dit niet gebeuren, dan raakt de processor letterlijk "de draad kwijt" !

Waarom verlangt het ene commando dus een RTS aan het eind en het andere niet? Antwoord: omdat het ene programma wordt aangeroepen met een JSR en het andere met een JMP!

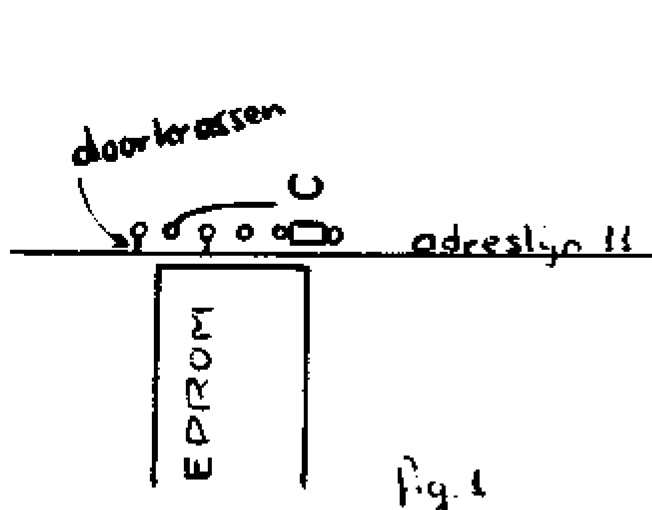
Ton Otten.

Het is al meerdere malen vermeld dat de eprom-voetjes op de schakelkaart, die gebouwd zijn voor de 2532, omgebouwd kunnen worden voor 2732. Hoe dat precies in z'n werk gaat, zal ik bij deze proberen uit te leggen.

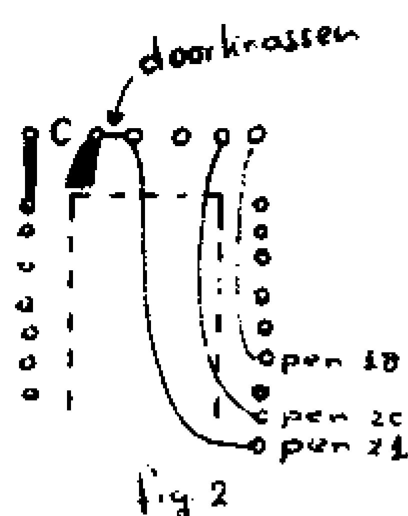
Naast de voetjes A,2 t.e.m. A,7 zit aan de kant van de pootjes 12 en 13 een condensator (C). Naast die condensator zitten 4 gaatjes die ik voor het gemak zal nummeren van 1 tot 4 (nr.1 het dichtst bij de C)

Bekijken we eerst de componentzijde (fig.1). De gaatjes 2 en 4 zijn verbonden met adreslijn A11. De verbinding tussen gaatje 4 en A11 doorkrassen.

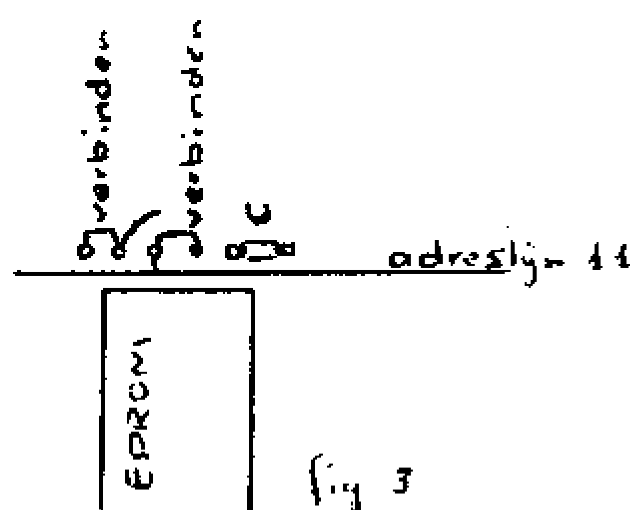
Aan de soldeerzijde (fig.2) zien we dat gaatje 1 verbonden is met de + van de condensator. Deze verbinding ook door-krassen.



COMPONENTZIJDE



SOLDEERZIJDE



COMPONENTZIJDE

Daarna worden de gaatjes 1 en 2 met elkaar verbonden en ook de gaatjes 3 en 4 met elkaar (fig.3).

Het betreffende voetje is nu geschikt voor een 2732-eprom.

De situatie bij de voetjes A,2 en A,7 is qua layout een ietsje anders dan de tekeningen aangeven. De ombouw gaat echter analoog aan de beschrijving en zal geen problemen opleveren, denk ik.

Bovenstaand verhaal geldt voor elk van de 6 voeten en hoewel "terugombouw" niet echt moeilijk is, is het misschien raadzaam om niet meteen alle voeten om te bouwen, doch slechts die welke u nodig heeft.

Vergewis u er terdege van dat doorgekraste printsporen ook echt dóórgekrast zijn. Eproms zijn i.h.a. bijzonder gevoelig voor verkeerde aansluitingen.

Verwijder ook voor alle zekerheid de eproms voordat u soldeert of de ohmmeter gebruikt.

Bram Post

```
>LIST
   10 DIM LL13,T64;C=0
   20 !#2800=#326A3173;!#2804=#23272F3F;!#2808=#38302021
   30 !#280C=#01013E3C;!#2810=#4C010101;!#2814=#1A180501
   40 !#2818=#0E12020C;!#281C=#0D170410;!#2820=#0F060714
   50 !#2824=#030A1D16;!#2828=#0B110903;!#282C=#201C1B19
   60 FOR A=1TO2;DIM P(-1)
   70 P.$21
   80[
   90:LL0 LDA#80
  100:LL1 CMP@#20;BEQLL11;CMP@#2C;BCCLL6;CMP@#5B;BCSLL6
  110 TAX;LDA#2800-#2C,X;LDY@8;STY#81
  120:LL2 ASLA;DEC#81;BCCLL2;STA#83
  130:LL3 LDA#83;ASLA;STA#83;LDY@1;BCCLL4;LDY@3
  140:LL4 LDA@#40;STA#B80B;LDA@#C0;STA#B80E
  150 LDA@#50;STA#B804;LDA@2;STA#B805;STA#B807;CLI
  160 JSRLL7;SEI;LDY@1;JSRLL7;DEC#81;BNELL3
  170:LL5 LDY@2;JSRLL7
  180:LL6 RTS
  190:LL7 TYA;ASLA;ASLA;TAY
  200:LL8 LDA#82;:LL9 LDX@#FA;:LL10 DEX
  210 BNELL10;SEC;SBC@1;BNELL9;DEY;BNELL8;RTS
  220:LL11 LDY@7;JSRLL7;RTS
  230:LL12 LDA#B804;LDA#B002;EOR@4;STA#B002;PLA;RTI
  240]
  250 NEXTA;P.$6
  260 GOS.520
  270 P.$12;IN."SNELHEID "S
  280 FOR N=0TO (A.R.%65)
  290 N?T=A.R.%26+65;N.
  300 N?T=#D;?#82=S;P.$12
  310 P.$21'';?#E1=0
  320 FOR N=0TO(LENT-1)
  330 ?#80=N?T
  340 P.$(?#80)" "
  350 ?#204=LL12%256;?#205=LL12/256
  360 LINK LL0;NEXT;P.$6
  370 P.''"WILT U DE TEKST NOG EENS HOREN"'
  380 IN.'"(JA=1 NEE=2)"W
  390 IF W=1 GOTO 310
  400 P.'"WILT U DE TEKST ZIEN"'
  410 IN."(JA=1 NEE=2)"C
  420 IF C=1 P.$6;GOTO 320
  430 P."WILT U STOPPEN OF EEN NIEUWE"'
  440 P."TEKST HOREN"'
  450 P."STOPPEN=1"'"NIEUWE TEKST=2"'
  460 IN.W;IF W=2 GOTO 270
  470 P.$12''''"IK HOOP DAT U ER VEEL PLEZIER"'
  480 P."MEE HAD EN DAT U ER IETS"'
  490 P."VAN GELEERD HEEFT."'
  500 P.''"                    F.MEPSCHEN."'
  510 ?#E1=128;END
  520 P.$12"          morsetrainer"''''
  530 P."DIT PROGRAMMA GENEREERT EEN"'
  540 P."TEKST VAN MAXIMAAL 64 TEKENS"'
  550 P."DAARNA HOORT U DEZE TEKENS"'
  560 P."IN MORSE EN WORDEN EVENTUEEL"'
  570 P."EEN VOOR EEN UITGEPRINT."'
  580 P."DE SNELHEID VAN ZENDEN IS"'
  590 P."VARIABEL."'
  600 P."BESTE WAARDEN TUSSEN 15 EN 30"'''
  610 P."                    VEEL SUCCES!"'
  620 LINK#FFE3;P.$12;RETURN
```

Standaardisatie Zero Page gebruik.

Ik constateer de laatste tijd steeds vaker dat iedereen zo zijn eigen ideeën heeft over het gebruik van de zero page geheugens.
Tot voor kort moest ik routinetjes uit tijdschriften herschrijven van #8x naar #9x vanwege de PPT. Na verloop van tijd kwamen er steeds meer programma's die naar #9x geschreven werden. Dat wordt dus dringen in dat gebied, zodat ik deze meestal weer terugschrijf naar #8x, omdat ik de PPT toch niet meer gebruik.
Verder is er een tendens om de adressen #B0-#BF te gebruiken (DOS werkruimte). Kijk maar naar de assemblercursus en de schakelprogramma's van Bouma. Er zijn zelfs Toolkits die deze ruimte opeisen (jazeker!).
Als ik dus een ZP-byte nodig heb voor permanent gebruik (in de sfeer van b.v. #12 of #E7) kan ik door dit soort grappen van de 64 bytes tussen #80-#BF er geen een uitkiezen.
Nu zijn bronprogramma's meestal wel aan te passen en zeker als ze met een offset werken, maar een toolkit van 4k disassembleren pas ik voor.
Om deze soesa enigzins te beperken in de toekomst, stel ik voor om enige richtlijnen op te stellen t.a.v. het zero page gebruik.
Laten we als basis de monitor van Heuvel en de Jos-box nemen. De monitor gebruikt #90-#A9 en de Jos-box #90-#99 (van de "vrije" ruimte #80-#AF). Dit laat zich uitstekend combineren.
Ik kan natuurlijk niemand voorschrijven hoe hij zijn zero page indeelt, maar t.a.v. publikaties zouden we misschien een protocol kunnen opstellen. Ter inleiding:
- Routines die alleen tijdens de uitvoering een paar bytes nodig hebben, kunnen daarvoor #70-#7F nemen. Dit gebied wordt bij FP berekeningen als kladbloc gebruikt en is tijdens integer-routines dus vrij. Een voorbeeld zijn de inverteerroutines uit AN 8 '82 p.43. Deze zouden naar b.v. #70,#71 geschreven kunnen worden.
"Mirakel" van Van Hoesel (AN 8 '82 p.52) gebruikt #7x.
Ook de voorbeelden van de assemblercursus kunnen heel goed #7x gebruiken (voor de rest overigens alle lof voor deze cursus).
- Moeilijker wordt het bij routines die hun data wat langer nodig hebben. Een voorbeeld hiervan is de lees-routine voor joysticks (AN 8 '82 p.50). Als je hier #7x gebruikt, is de kans groot dat na een FP operatie de leesresultaten verloren zijn.
Als je in een programma geen Jos-box of monitor aanspreekt, is #9x beschikbaar.
Een STOP-CONTINUE-constructie valt ook in deze groep.
- Er zijn ook routines die een ZP byte ten allen tijde in beslag nemen (b.v. ALLCOS, AN 1 '83 p.59). Het byte #AB mag in dit geval voor niets anders gebruikt worden. Heb je geen DOS, dan kun je daarvoor #Bx nemen.

Samengevat stel ik voor:

#70 - #7F: voor gebruik in een enkele routine of statement
#80 - #8F: ieder idee is welkom
#90 - #A9: voor monitor, Jos-box en ander tijdelijk gebruik
#AA - #AF: zie verderop
#B0 - #BF: alleen voor persoonlijk gebruik;
in publikaties alleen met "waarschuwing"

#AA gebruik ikzelf voor een ASCII-dump in de monitor en voor #AB-#AF zie ik een mooie toepassing voor een STOP-CONTINUE-constructie. Het kopiëren van de geheugens 5,6,#13,#14 en #15 tijdens BASIC naar #AB-#AF biedt de mogelijkheid om dat programma na b.v. opvragen van waarden van variabelen weer door te starten door de locaties terug te kopiëren.

Sowieso is het een goede gewoonte om de gebruikte adressen in routines met een offset te programmeren met daarbij een verklaring (zie b.v. ALLCOS). Dan kan een ieder zijn eigen aanpassingen maken.

Op- en vooral aanmerkingen op dit verhaal zie ik volgaarne tegemoet (maar dan wel via AcornNieuws!).

Bram Post

LICHTKRANT

```
 10REM  LICHTKRANT OF MONITOR.
 20REM  MAX.64 KARAKTERS.
 30REM  MODE1/MODE2/MODE3.
 40REM  NA TEKST <RETURN>.
 50REM  STOPPEN DOOR <SHIFT>.
 60REM
 70REM  JOOST DE WIJS,
 80REM  VELDHOVEN
 90REM
100GRMOD;DIMA64;?225=0;P."UW TEKST:"'
110IN.$A;P.''"UW TEKST WORDT INGELEZEN."
120A=#81C1;B=#3A00
130aF.I=0TO224S.32
140?B=I?A
150B=B+1;IFB=#3C00;G.200
160N.
170A=A+1;IFA=#81E0;A=#8340
180IFA=#8360;A=#84C0
190G.a
200DIMLL9;F.I=0TO9;LL(I)=-1;N.
210F.I=1TO2;P=#2800;P.$21;[
220:LL0LDA@0;STA#90
230LDA@#3A;STA#91
240:LL1LDA#B001;AND@#80;BNELL2;RTS
250:LL2LDX@0;LDY@0
260:LL3LDA(#90),Y;STA#39F8,X;JSR#FE6B
270INC#90;BNELL4;INC#91
280:LL4INX;CPX@8;BNELL3
290:LL5LDX@0
300:LL6ASL#39F8,X;ROL#99
310INX;CPX@8;BNELL6
320JMPLL8
330:LL7INY;CPY@8;BNELL5
340LDA#91;CMP@#3C;BNELL1
350JMPLL0
360:LL8LDX@15
370:LL9ROL#99;ROL#99
380ROL#8180,X;ROL#99
390ROL#8190,X;ROL#99
400ROL#81A0,X;ROL#99
410ROL#81B0,X;ROL#99
420ROL#81C0,X;ROL#99
430ROL#81D0,X;ROL#99
440ROL#81E0,X;ROL#99
450DEX;BPLLL9
460JSR#FE66;JMPLL7
470]
480P.$6;N.
490CLEAR1;LINK#2800
500TXMOD;END
```

ASSEMBLER DEEL **6**

DE STACK

De stack (spreek uit: stek) is een stukje RAM geheugen dat door de microprocessor wordt gebruikt. De naam 'stek' zal de bouwkunde-electronici (bestaan die?) bekend in de oren klinken. De houtstek is een stapel hout waar iets opgegooid kan worden en er ook weer afgehaald kan worden. De uitdrukking 'gooi het maar op de stek' komt precies overeen met wat de microprocessor doet, alleen iets beschaafder.
Bij een 6502 microprocessor ligt de ruimte waar de stack zich bevindt altijd vast. De ruimte ligt vast van adres #0100 - #01FF. Vandaar dat bij iedere 6502 computer hier RAM geheugen zit.
De stack kan worden voorgesteld als een boek met 255 bladzijden. Iedere bladzijde stelt 1 byte voor. Als iets op de stack gezet wordt, is dit altijd van acheraf, dus de microprocessor begint te schrijven op blz. 255 en gaat zo naar voren. (254, 253 enz.)
De 6502 moet bijhouden waar voor het laatst iets is neer gezet. Hiervoor is, om het voorbeeld van het boek nog maar even te gebruiken, een soort bladwijzer aanwezig. Deze bladwijzer heet de stackpointer.
Het hoogste byte van de stackpointer is altijd #01, want de stack zelf liep immers van #0100 tot #01FF. Alleen het low-byte van de stackpointer kan iedere waarde aannemen. Vandaar dat de stackpointer een 1 bytes register is, want de stackpointer 'weet' dat het high-byte altijd #01 is.

Wat gebeurt er als b.v. 2 getallen op de stack gezet worden ?

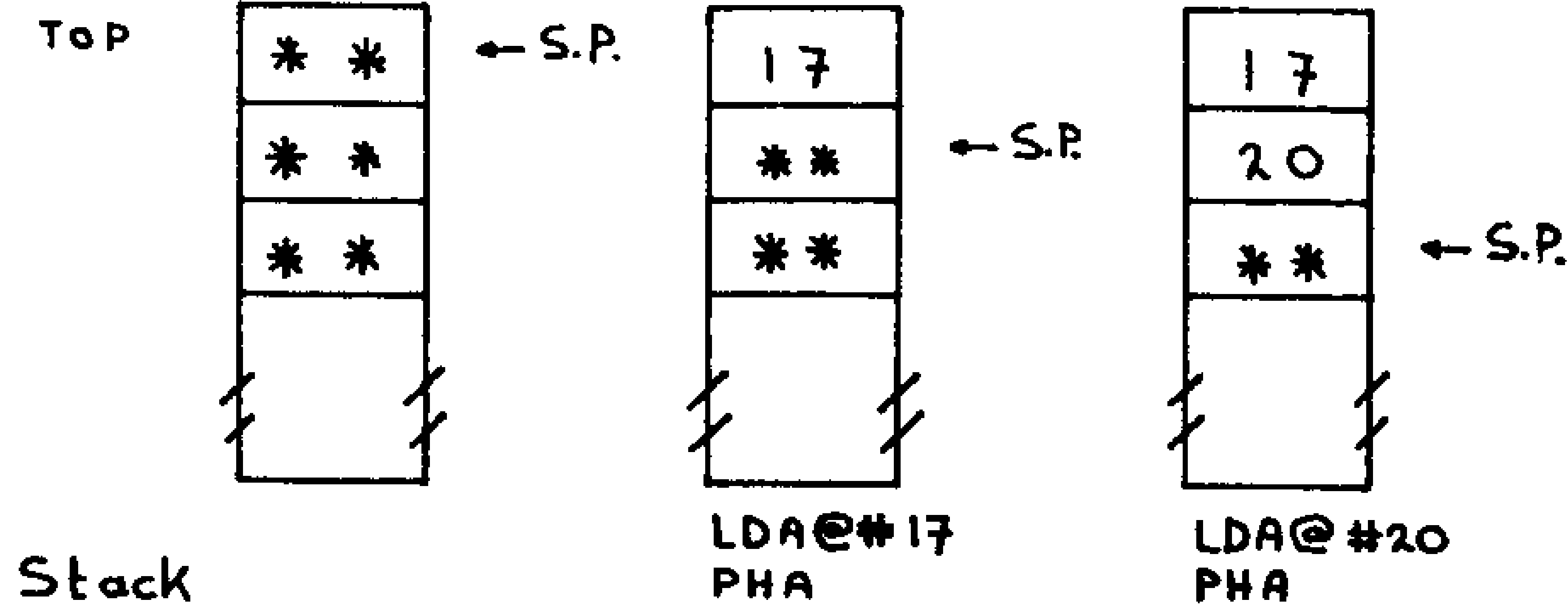
Wel, stel dat de stack 'leeg' is, (bijvoorbeeld na een 'BREAK'), dus de stackpointer staat op #(01)FF

* Het eerste getal gaat op de stack
* Vanwegê bovenstaande aktie wordt de S.P. met 1 verlaagd
* De StackPointer bevat nu de waarde #FE
* Het tweede getal gaat op de stack
* Weer als gevolg hiervan wordt de S.P. met 1 verlaagd
* Enz.

(zie ook de tekening)

Waar gebruikt de microprocessor de stack zoal voor?

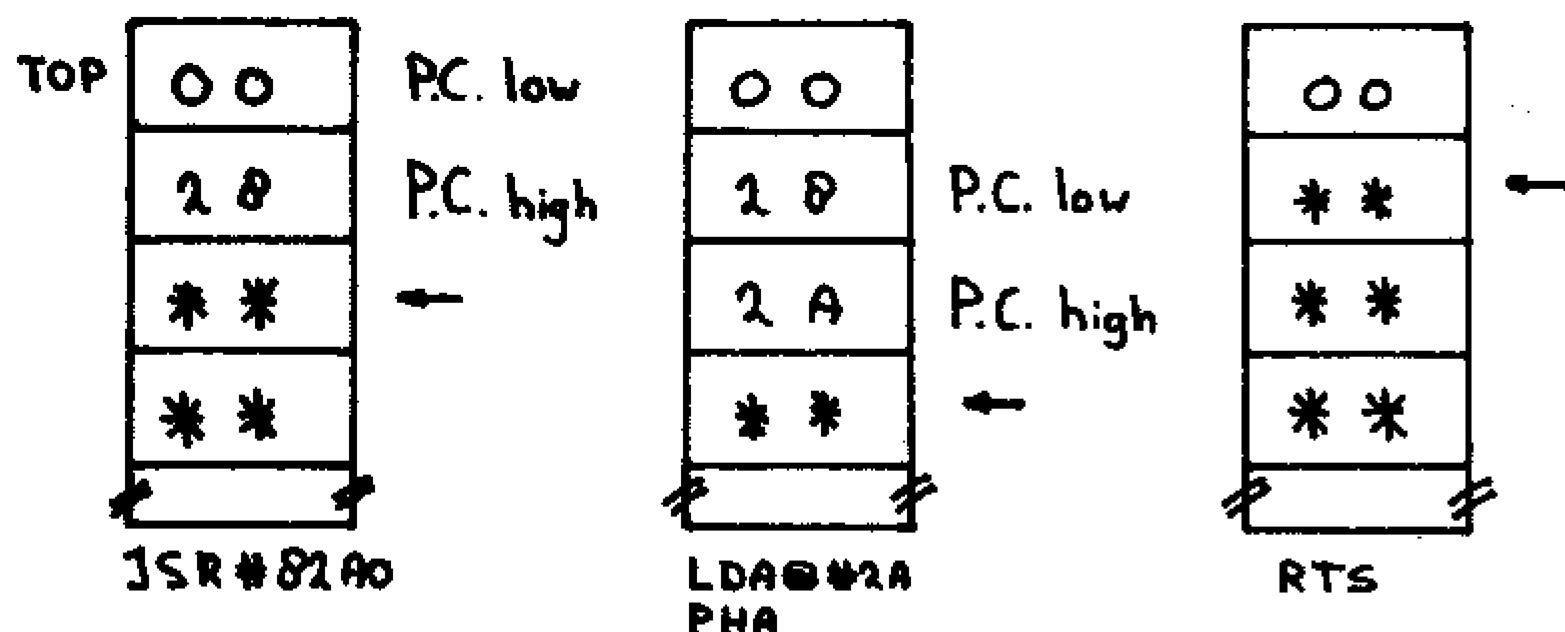
Als de microprocessor bijvoorbeeld opdracht krijgt om naar een subroutine te springen, dan zal de microprocessor het adress moeten onthouden waar hij moet terugkeren als de subroutine voltooid is. Dit 'onthouden' gebeurd op de stack. Het neemt twee plaatsen van de stack in beslag, namelijk het low byte en het high byte van de program counter.



Iets over de Atom en de 6502

255 bytes stack ruimte is heel erg veel. Veel meer dan zelfs een proffesioneel systeem ooit nodig zal hebben. Vandaar dat de software ontwikkelaars van de Acorn Atom de helft van de beschikbare stack ruimte hebben 'afgesnoept'. De ruimte van #0100 tot #013F wordt b.v. als inputbuffer gebruikt (als een opdracht of statement wordt ingetikt, wordt het hier neergezet en na een bevestiging met de RETURN toets wordt op dit adres begonnen met interpreteren) De ruimte van #0140 tot #017F wordt gebruikt voor string afwerking. Hieruit volgt dat er effectieve stackruimte overblijft van #0180 tot #01FF. Dit is evengoed nog erg veel. Neem dat maar aan.
Ook de programmeur kan gebruik maken van de stack. Hier voor zijn de volgende instrukties: PHA, PLA PHP en PLP.
Met de PHA instruktie kan de inhoud van de Accu op de stack gezet worden. De stackpointer wordt automatich met 1 verlaagd. PHA betekent PusH Accu, vrij vertaald: duw de accu (op de stack)
Met de PLA instruktie wordt de waarde weer van de stack afgehaald en in de accu gestopt. De stackpointer wordt nu (logisch) met 1 verhoogd. PLA betekent PulL Accu (sleur ervan af)
Met de PHP kan de inhoud van het vlaggenregister op de stack gezet worden. Er wordt dus een kopie gemaakt van het status register en dat verdwijnt naar de stack. PHP = Push Processor status
Met PLP is het weer mogelijk om de waarde weer van de stack af te halen en het in het vlaggenregister te plaatsen.
Het grote voordeel van een push instruktie is, dat de programmeur zich niet hoeft druk te maken waar een bepaalde waarde is neergezet. Het is ook mogelijk om de X en Y registers op de stack te zetten. Dit kan niet in een keer. Eerst zal de waarde van het X of Y register naar de Accu moeten worden gekopieerd. Bijvoorbeeld zo: TXA, PHA TYA, PHA. Beide registers staan nu op de stack.

Er moet wel voor gezorgd worden, dat er netzoveel van de stack moet worden afgehaald, als dat er opgezet is. Dit lijkt ingewikkelder dan het is. Zoals verteld, maakt de microprocessor b.v. gebruik van de stack bij het springen naar en subroutine. Voorbeeld: Stel dat vanaf adres #2800 naar een subroutine op adress #82A0 is gesprongen. (JSR #82A0) Op de stack staat nu in volgorde 00 28. Wordt nu bijvoorbeeld in de subroutine met een PHA instruktie #2A gezet en er niet vanaf gehaald, dan zal bij een return- subroutine instruktie (RTS) terug worden gesprongen naar adress #2A28, want de processor pakt twee waardes van de stack. Op de stack stond: 00 28 2A. De twee laatste getallen worden er afgenomen. Zie tekening.



Een programma zal deze wat droge aflevering wat illustreren. Het programma stelt weinig voor, maar het maakt een duidelijk zichtbaar gebruik van de stack. Het programma geeft, na ge 'run'd te zijn, een kort piepje als er een toets wordt ingedrukt. Zo wordt een indruk van een 'zeer dure en proffesionele terminal' verkregen. Een 'BREAK' legt het piepje weer het zwijgen op.

```
 10 REM TOETSPIEPER
 20 DIM LL3
 30 P=#28A0;[
 40 JSR #FE94     HAAL TOETS BINNEN
 50 PHA           ZET ACCU OP DE STACK
 60 TYA           KOPIE VAN Y NAAR A
 70 PHA           ZET ( Y ) OP DE STACK
 80 TXA           IDEM VOOR X
 90 PHA            "    "
100 LDX @#30      BLEEP ROUTINE
110:LL2 LDY @#20
120 LDA #B002     SPEAKER
130 EOR @4        TOGGLE SPEAKER
140 STA #B002
150:LL3 DEY
160 BNE LL3
170 DEX
180 BNE LL2
190 PLA           HAAL ( X ) VAN DE STACK
200 TAX           KOPIE VAN A NAAR X
```

```
210 PLA          IDEM VOOR ( Y)
220 TAY            "     "
230 PLA          HAAL ACCU VAN STACK
240 RTS;]
250 REM VERZET DE LEESVECTOR
260 REM ZIE ATOM MANUAL
270 ?#20A=#A0;?#20B=#28
280 END
```

De stackpointer kan door de programmeur zelf worden beinvloed. Dit komt niet zo vaak voor. Tijdens b.v. een RESET van de 6502 gebeurt dit wel, want er moet met een schone lei worden begonnen. Er zijn twee stackpointer bewerkings instrukties: TSX en TXS. TXS verplaatst het low byte van de S.P. naar de waarde van het X register. TSX verplaatst de waarde van de stackpointer naar het X register.

Tot slot:

De stackpointer wordt in Atom basic nog al eens gereset. (op #FF gezet). Pas dus op met gecombineerde basic en assembler routines.

*** ** ** ***

* DE STACK LOOPT VAN #0180 - #01FF
* PHA EN PLA DOEN ACCU OP / VAN DE STACK
* PHP EN PLP DOEN VLAGGEN REGISTER OP EN VAN DE STACK
* 3 GETALLEN OP DE STACK? DAN OOK 3 GETALLEN VAN DE STACK !!
* OPPASSEN MET DE STACK POINTER

LEENDERT BIJNAGTE

```
>LIST; CASSETTE-CATALOG
   10 DIM PP4
   20 F.I=0TO1;P=#2800;[
   30:PP0 JSR#FB8E;BVSPP1;RTS
   40:PP1 BEQPP2;LDY@0;JSR#F999
   50 JSR#F7EC;BNEPP3
   60:PP2 JSR#FBC9;JSR#FBE2
   70 LDA#D9;BEQPP4;LDA#DB;BMIPP0
   80:PP4 JSR#F992;JSR#F7EC;ROL#DB
   90 BPLPP3;INX;JSR#F7F1
  100 LDA#FD,X;JSR#F802
  110:PP3 JSR#FFED;BNEPP0
  120];N.
  130 IN."NAAM BAND"$P
  140 ?#FE=0
  145 REM PRINTER-BESTURINGEN VOOR MICROLINE
  150 P.$2$27$66$31'''$P''$30$3$12
  160 ?#FE=10
  170 LI.#FC3E
  180 P.$2;LI.PP0;P.$3;E.
```

REM Dit programma maakt een
REM katalogus van een band,
REM net zoals *CAT, met dit
REM verschil dat nu alleen het
REM EERSTE en het LAATSTE blok
REM van elke file wordt geprint.

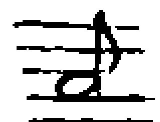
```
10 DIM LL(32),λ(64)
20 K=#8000;L=#9000;M=#8100;N=#9100
30 J=#8F00
40 FOR Q=1TO2
50 P.$21;DIM P(-1)
60[
70:LL0 LDX@0;LDA@#20
80:LL1 STAJ,X;INX;BNELL1
90:LL2 STAL,X;INX;BNELL2
100:LL3 STAN,X;INX;BNELL3
110 RTS
120:LL4 LDX@0
130:LL5 LDAK,X;STAL,X
140 INX;BNELL5
150:LL6 LDAM,X;STAN,X
160 INX;BNELL6
170 RTS
180:LL9 LDX@0;STX#80
190 LDA@#0F;STA#81
200:LL10 LDAL-#21,X;CMP#81;BNELL11;INC#80
210:LL11 LDAL-#20,X;CMP#81;BNELL12;INC#80
220:LL12 LDAL-#1F,X;CMP#81;BNELL13;INC#80
230:LL13 LDAL-1,X;CMP#81;BNELL14;INC#80
240:LL14 LDAL+1,X;CMP#81;BNELL15;INC#80
250:LL15 LDAL+#1F,X;CMP#81;BNELL16;INC#80
260:LL16 LDAL+#20,X;CMP#81;BNELL17;INC#80
270:LL17 LDAL+#21,X;CMP#81;BNELL18;INC#80
280:LL18 LDA#80;CMP@2;BEQLL20
290 LDA#80;CMP@3;BEQLL19
300 LDA@#20;STAK,X;JMPLL20
310:LL19 LDA@#0F;STAK,X
320:LL20 LDA@0;STA#80;INX;BNELL10
330:LL21 LDAN-#21,X;CMP#81;BNELL22;INC#80
340:LL22 LDAN-#20,X;CMP#81;BNELL23;INC#80
350:LL23 LDAN-#1F,X;CMP#81;BNELL24;INC#80
360:LL24 LDAN-1,X;CMP#81;BNELL25;INC#80
370:LL25 LDAN+1,X;CMP#81;BNELL26;INC#80
380:LL26 LDAN+#1F,X;CMP#81;BNELL27;INC#80
390:LL27 LDAN+#20,X;CMP#81;BNELL28;INC#80
400:LL28 LDAN+#21,X;CMP#81;BNELL29;INC#80
410:LL29 LDA#80;CMP@2;BEQLL31
420 LDA#80;CMP@3;BEQLL30
430 LDA@#20;STAM,X;BNELL31
440:LL30 LDA@#0F;STAM,X
450:LL31 LDA@0;STA#80;INX;BNELL21
460 RTS
470]
480 NEXT Q
490 P.$6
500 P.$12"                    life"'
510 P."M.B.V. DE CURSOR BESTURING"'
520 P."TYPT U ERGENS OP HET SCHERM"'
530 P."EEN AANTAL ""O""'S"'
540 P."DE FIGUUR VERANDERT NU STEEDS"'
550 P."VOLGENS EEN VAST PATROON"'
560 P."DE SNELHEID HIERVAN IS VARIABEL"'
570 IN.'"UW KEUZE (1-10)"S
580 P.$12''''''''
590 LINK LL0
600 IN.$X
610 LINK LL4
620 LINK LL9
630 LINK LL4
640 FOR W=1TO((10-S)*40);WAIT;N.
650 GOTO 620
660 END
```

Ooooh!

Wat ben

je

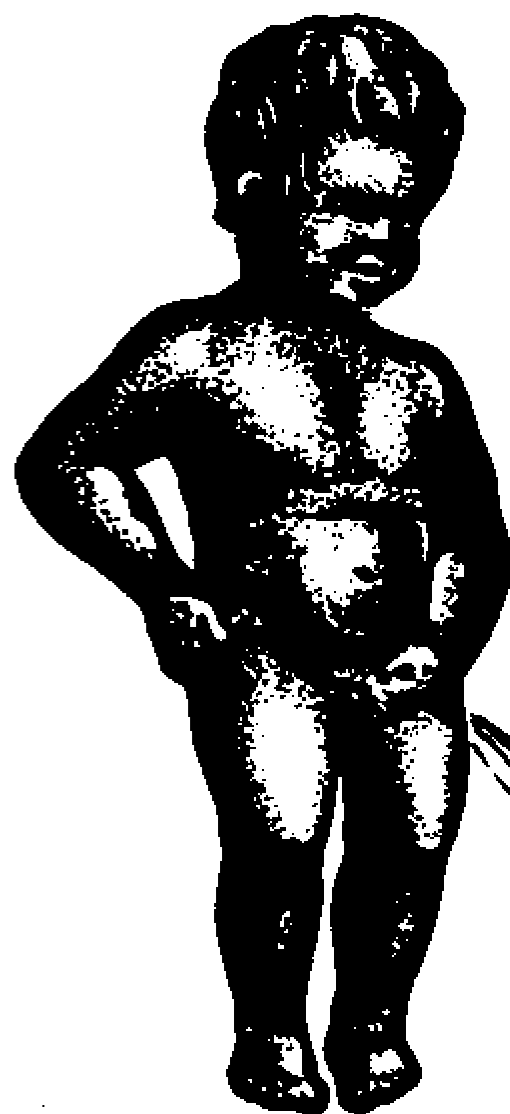
mooooi



HIER GAAT HET OM!! !!!!

```
10G.1200
20***berekening van PI tot***
30***  op 1000 decimalen  ***
40
50  DOOR: GEORGES CARON
60     51 RUE SERAPHIN CORDIER
70     62260, AUCHEL
80     FRANKRIJK.
90
100 VOOR ACORN ATOM BEWERKT
120 DOOR: JJH WOLDRINGH
130    BOTTEMAHEERD 4
140    9737 NE/ GRONINGEN
150    TEL. 050-412929
160

1000P.$12,"berekening van PI."'
1010F=1000;H=100;Z=4;?#E1=0
1020IN."HOEVEEL DECIMALEN(1000): ",M
1030M=(M+1)/3+1;DIM AAM
1035D=0;F.I=1 TO M;AAI=1;N.
1040O=%(M*3*LOG10/LOG2)-1
1100F.N=O TO 1 STEP -1
1110  AA1=AA1+H;P=2*N+1
1120  F.I=1 TO M
1130    D=D*F+AAI*N
1140    AAI=D/P
1150    D=D-AAI*P
1160  N.
1170  D=0;P.N
1180N.
1200F.I=M TO1 STEP -1
1210  IF AAI)=F AA(I-1)=AA(I-1)+1;AAI=AAI-F
1220N.
1230IF M)160 P.'"ZET PRINTER AAN "'$2
1300P.'"HIER VOLGT HET GETAL PI"'"MET ",3*M-1," DECIMALEN:"'
1310@=0;AA1=AA1-100
1320P."3. ",AA1
1330F.I=2 TO M
1340  IF AAI(100 P."0"
1345  IF AAI(10  P."0"
1350  P.AAI
1360N.
1370P.'$3;?#E1=128;E.
>
>
>
```



UIT: "HOBBYSCOOP"

REM: Volgens "Hobbyscoop" heeft dit programma ca. 12 uur nodig; Onze Acorn Atom doet 't in 4 uur!!!

UITSPRAAK van een TH-prof (die zo'n 50 decimalen van PI uit z'n hoofd kende):

"Op de TH werken we niet met graden, daar werken we met PI's!!!"

15. DE CASSETTE INTERFACE

ATOM FORTH heeft een eigen cassette interface. Hiermee is het mogelijk geschreven programma's op cassette te zetten, en ze later weer in te lezen. Een programma wordt verdeeld in schermen. Elk scherm kan 8 regels van 64 karakters bevatten (totaal 512 bytes). In zo'n scherm staat het programma, zoals het vanaf een toetsenbord zou worden ingetypt, en dus niet een reeds gecompileerde versie. Men schrijft zijn programma met behulp van een editor in een scherm. De inhoud daarvan kan vervolgens gecompileerd, getest en eventueel verbeterd worden. Pas zodra het correct werkt zullen we het op cassette bewaren. Vervolgens gaan we dan verder met het volgende scherm, totdat het totale programma in een aantal opeenvolgende schermen is bewaard.

Het commando om een of meerdere schermen te laden heet LOAD. Het verwacht dat er een nummer op de stack staat, dat aangeeft welk scherm geladen moet worden. Men kan zijn programma's dus alleen nummers geven en geen naam. De schermen mogen genummerd worden van 0 tot 999. Om zelf programma's te schrijven in schermen is het noodzakelijk de beschikking te hebben over een editor. ATOM FORTH heeft zo'n editor. Deze moet worden geladen door:

```
6  LOAD
```

De computer zal nu reageren met:

```
>6 PLAY TAPE
```

We kunnen nu de cassette starten en vervolgens op de spatiebalk drukken. De schermen 6 tot en met 14 worden nu geladen. Na elk scherm dat geladen is zal de computer de inhoud daarvan meteen compileren. Vervolgens verschijnt er op het beeldscherm het nummer van het volgende te laden scherm. Nadat alle schermen geladen zijn, beschikken we over een complete editor en cassette interface. We kunnen de inhoud van een scherm zichtbaar maken met behulp van het commando LIST. Ook dit commando verwacht dat op de stack het nummer van het gewenste scherm staat. Probeer het volgende maar eens

```
14 LIST
```

Er volgt nu een listing van scherm 14. Dit was het laatste scherm, dat we zojuist hebben geladen. Als we hadden getypt

```
7 LIST
```

dan zou de computer hebben gereageerd met

```
>7 PLAY TAPE
```

De computer moet scherm 7 eerst weer laden in zijn geheugen. De schermen worden bij het laden in de tape-buffer gezet.

We kunnen zelf de opdracht geven om het in de tape-buffer aanwezige scherm te compileren. We gebruiken daarvoor het commando:

ENTER

We zullen dit woord voornamelijk gebruiken in de fase, waarin het programma nog geschreven wordt. Via de editor schrijven we dan een scherm vol. Met ENTER compileren we dit scherm. Vervolgens testen we het en veranderen wat nodig is. Zodra het werkt, kunnen we het op cassette zetten. Daarvoor staat het commando

SAVE

tot onze beschikking. De computer zal reageren met

RECORD TAPE

Na een druk op de spatiebalk zal het programma worden opgenomen op de cassette. Zodra de computer klaar is verschijnt de melding

STOP TAPE

ten teken dat de computer klaar is.

We hebben al gezien dat na een LOAD opdracht de computer meer dan een scherm achter elkaar kan laden. Door aan het einde van een scherm op te nemen het woord

-->

geven we aan dat er nog een volgend scherm geladen moet worden. De schermen moeten opeenvolgend genummerd zijn!
Als --> niet is gebruikt aan het einde van een scherm, dan stopt het laden en compileren aan het einde van dit scherm of bij het woord

;S

Bij dit woord stopt het interpreteren, zodat de rest van het scherm commentaar mag bevatten. We kunnen ook op andere plaatsen op het scherm commentaar opnemen, maar dat moet dan tussen haakjes geplaatst worden.

Het nummer van het huidige, in de tape-buffer aanwezige, scherm, wordt bewaard in de variabele SCR .
We kunnen dus een listing opvragen van dit scherm door te typen:

SCR @ LIST

We zullen zien dat de EDITOR van ATOM FORTH daarvoor een korter commando kent.

16. DE EDITOR

Als we willen beginnen met het programmeren van een programma, dan zal eerst de tape-buffer moeten worden leeg gemaakt, we moeten een nummer geven aan het scherm, we moeten dit scherm laten zien en vervolgens moeten we nog aangeven dat we editor-commando's willen gebruiken. (Waarom we dit moeten aangeven, zal een volgende keer worden behandeld.) Gelukkig kan dit alles in ATOM FORTH met een commando, nl:

PROGRAM

De computer zal dan vragen:

FIRST SCREEN NUMBER ?

Nadat we een nummer (0-999) hebben ingetypt, zal er een scherm met acht genummerde blanko regels verschijnen. We kunnen nu beginnen met het programmeren van ons programma. Bijvoorbeeld

```
0 P  ( BEGROETING )
1 P  DECIMAL
2 P  : GROET ." HALLO, WELKOM BIJ ATOM FORTH. "
3 P          CR ( EINDE ) ;
```

Een listing van dit programma kunnen we krijgen door de opdracht

L

We kunnen dit programmaatje ook compileren en testen via

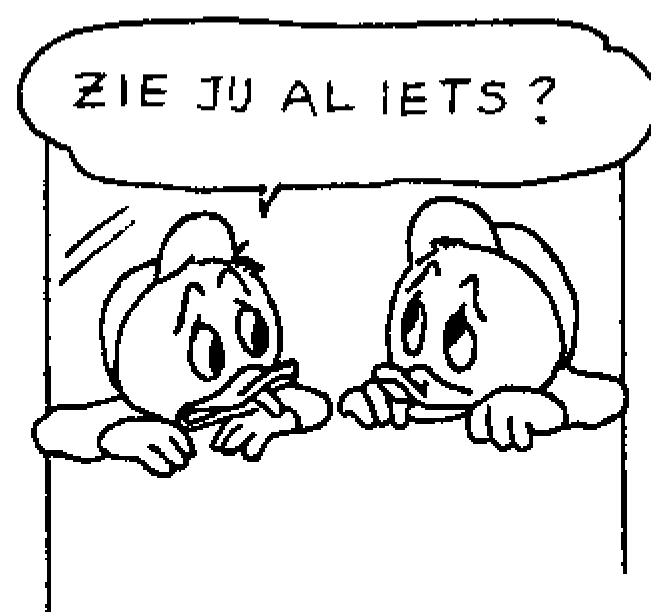
ENTER
GROET

We zien dat het commando P de daarop volgende tekst zet op de gegeven regel, en dat het commando L een listing geeft van het scherm. De EDITOR kent de volgende commando's:

P   Zet de daarop volgende tekst op de gegeven regel.
D   Verwijdert de regel, en bewaart hem op PAD. Schuift alle volgende regels een plaats naar boven.
E   Maakt de regel schoon.
H   Zet de inhoud van de regel op PAD.
I   Zet de inhoud van PAD op de gegeven regel. Alle volgende regels schuiven een plaats naar beneden.
R   Vult de regel met de inhoud van PAD.
S   Voegt een blanko regel tussen.
T   Toont de inhoud van de regel, en bewaart deze inhoud ook in PAD.

We zien dat de EDITOR een soort buffertje gebruikt, waar tijdelijk de inhoud van een regel kan worden opgeslagen. We noemen het beginadres van dit buffertje PAD.
Oefen zelf eens met de EDITOR.   TOT DE VOLGENDE KEER!

```
>LIST
   10 REM GRAPHICWINDOW
   20 REM
   30 REM AUTEUR: GERT DE JAGER
   40 REM               PAPENDRECHT
   50 REM
   60 REM               regio rotterdam
   70 REM
   80 REM ASSEMBLER
   90 DIM LL(1),P(-1)
  100 PRINT $21
  110 FOR A=0 TO 1
  120[
  130:LL0 LDA #5B \X COORDINAAT GROTER DAN 255
  140     BNE LL1 \JA? RTS
  150     LDA #5D \Y COORDINAAT GROTER DAN 191
  160     BNE LL1 \JA? RTS
  170     LDA #5A
  180     CMP #80 \X KLEINER DAN ONDERSTE X-GRENS
  190     BCC LL1 \JA? RTS
  200     CMP #82 \X GROTER DAN BOVENSTE X-GRENS
  210     BCS LL1 \JA? RTS
  220     LDA #5C
  230     CMP #81 \Y KLEINER DAN ONDERSTE Y-GRENS
  240     BCC LL1 \JA? RTS
  250     CMP #83 \Y GROTER DAN BOVENSTE Y-GRENS
  260     BCS LL1 \JA? RTS
  270     JMP #F7AA \POINT PLOT ROUTINE (CLEAR 4)
  280:LL1 RTS
  290]
  300 NEXT A
  310 PRINT $6
  320 REM 4 COORDINATEN NODIG
  330 REM VOOR X: ONDER- EN BOVENGRENS
  340 REM VOOR Y: ONER- EN BOVENGRENS
  350 REM VOOR X: 0-255
  360 REM VOOR Y: 0-192
  370 REM ONDERGRENS MAG NIET GROTER
  380 REM ZIJN DAN BOVENGRENS
  390 REM VOORBEELD COORDINATEN
  400 REM X: BOVEN- EN ONDERGRENS
  410 ?#80=01;?#82=180
  420 REM Y: BOVEN- EN ONDERGRENS
  430 ?#81=01;?#83=180
  440 REM ZET GRAFISCHE MODE
  450 CLEAR 4
  460 REM ZET NIEUWE PLOTVECTOR
  470 ?#3FE=LL0%256;?#3FF=LL0/256
  480 REM VOORBEELDPROGRAMMA
  490 MOVE 128,96
  500 DRAW (ABSRND%256),(ABSRND%192)
  510 GOTO 490
```



JOYSTICK interface voor de atom

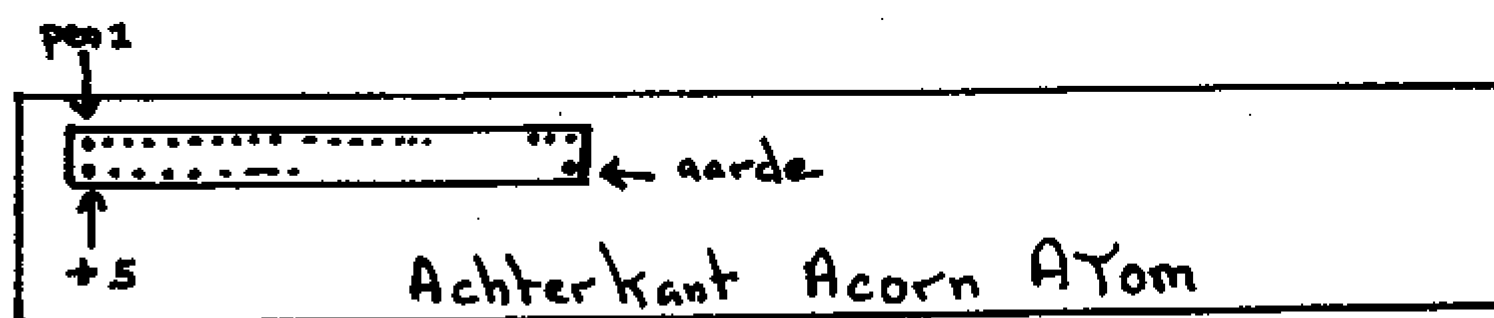
Deze interface zet de tijd die een bepaalde condensator nodig heeft om ontladen te worden,om in een getal,dat weggeschreven wordt op een zelf te bepalen geheugenplaats.De tijd,om te ontladen,en dus de waarde van dat getal wordt mede bepaald door de stand van de joystick.
In de joystick zitten 2 potmeters, waarvan er een dient voor de x-as en de andere voor de y-as.Bij een 1 Mhz clockfrequentie en een waarde van ongeveer 37 n voor timing capacitor a en b krijgt u getallen tussen 1 en 255.De waarde van de joystick potm. moet dan 50K zijn (telec f.17,85)
Voor andere getallen moet u de volgende formules gebruiken.
Pulsduur=gewenste getal x 9 us ; cond.waarde=puldduur/1,1 x de weerstand.
C in Farads, tijd in seconden, en weerstand in ohms.

TRIGGER FROM PROCESSOR OUTPUT STROBE
TIMING CAPACITOR A
+5V
4 RESET
6 TRIGGER
10 RESET
8 TRIGGER
1K
1K
2 DISCHARGE
2 THRESHOLD
TO PROCESSOR INPUT LINE "A"
5 OUTPUT
IC1 556
JOYSTICK POTENTIOMETER A
+5V
TO PROCESSOR INPUT LINE "B"
9 OUTPUT
14
JOYSTICK POTENTIOMETER B
0.01μF
13 DISCHARGE
12 THRESHOLD
TIMING CAPACITOR B
CONTROL VOLTAGE 3
CONTROL VOLTAGE 11
0.01μF
0.01μF

Number	Type	+5 V	GND
IC1	556	14	7

UIT: "BYTE", SEPT. 1980

AANSLUITING OP DE ACORN ATOM



De triggeraansluiting op PB 1 ,dit is connectorpen 9
Outputlijn 'a' op PB 2, dit is pen 8
Outputlijn 'b' op PB 3, dit is pen 7

Assembler routine voor het lezen van de joystick.

opm. voor twee joysticks heeft u ook twee keer de hardware nodig
en 3 poortlijnen extra.

```
:LL1  LDX @≠01      entry één joystick
      BNE  LL0      geforceerde jump
:LL2  LDX @≠03      entry voor twee joystick's
:LL0  LDA @≠02      initialiseer timer pointer
      STA  ≠B802
      PHP           interrupt status naar de stack
      SEI           disable interrupt
:LL3  ASL           stel timer pointer in
      LDY @0        trigger timer in VIA
      STY  ≠B802    laag naar
      LDY @2        hoog
      STY  ≠B802
      LDY @≠FF      initialiseer teller
:LL4  INY           verhoog teller
      BIT  ≠B802    test teller
      BNE  LL4      hoog? doorgaan met tellen.
      PHA
      TYA
      STA  ≠021C,X  vrije geheugenplaats
      DEX
      BPL  LL3  nog meer tellers
      PLP       nee, haal status van stack
      RTS
```

Link LL1 : X waarde joystick 1 komt op ≠21C
Y waarde " " " " ≠21D

Eventueel bij Link LL2 (voor twee joysticks)
X waarde joystick 2 op adres ≠21E
Y " " " " " ≠21F

Basic testprogramma:

```
10  LINK LL1
20  PRINT ?≠23C,?≠23D,';GOTO 10
```

u krijgt dan twee rijen getallen X en Y die wisselen met de stand van de analoge joystick.

Programma om lijnen op het scherm te tekenen:

```
10 CLEAR 4
20 DO  LINK LL1; MOVE ?≠21C,?≠21D
30 DRAW ?≠21C,?≠21D
40 UNTIL Ø
```

```
LIST
  10 !#20A=#F36EA141    } TOOLKIT AAN!
  20 !#202=#A000A1E2    }
  30 P.$12;@=0;L=1;O=0;B=0;?#E1=0
  40 DIM RR(13)
  50 RESTORE
  60 FOR N=1TO13
  70   READ RR(N)
  80 NEXT
  90 P."WAT VOOR WEERSTAND ZOEKT U?"'
 100 FIN."(WAARDE IN OHMS)"%R
 110 P.''"WAT IS DE TOLERANTIE "'
 120 FIN."(WAARDE IN % )"%T
 130 FIF %R<=100       F=1     ;G.a
 140 FIF %R<=1000      F=10    ;G.a
 150 FIF %R<=10000     F=100   ;G.a
 160 FIF %R<=100000    F=1000  ;G.a
 170 FIF %R<=1000000 F=10000;G.a
 180 P.'"WEERSTAND TE GROOT"$7$7$7$7$7$12;G.90
 190a GOS.t
 200 IF L=2 THEN %O=%R
 210 M=0;DO M=M+1
 220 FUNTIL RR(M)*F)=%R
 230hV=1;I=0
 240bI=I+1
 250 %P=(RRM*F*RRI*V)/(RRM*F+RRI*V)
 260 FIF %P)%O O=1
 270 FIF %P<%B B=1
 280 IF B=1 AND O=1 GOTO c
 290 B=0;O=0
 300 IF I<>13 GOTO b
 310 IF V<>100000 I=0;V=V*10;G.b
 320 IF M<>13 M=M+1;G.h
 330 %K=%T;%T=10;L=2;GOTO a
 340cIF L<>2 GOTO e
 350 %T=%K;GOS.t
 360 Z=1;U=0;O=0;B=0
 370dU=U+1
 380 %Q=(%P*RRU*Z)/(%P+RRU*Z)
 390 FIF %Q)%O O=1
 400 FIF %Q<%B B=1
 410 IF B=1 AND O=1 G.f
 420 B=0;O=0
 430 IF U<>13 GOTO d
 440 IF Z<>100000 U=0;Z=Z*10;G.d
 450 P.$12'''"WEERSTANDNIET GEVONDEN"';?#E1=0
 460 BEEP 20,20:END
 470eP.$12''"R1 = "RRM*F';?#E1=0
 480 P."R2 = "RRI*V''
 490 FP."R(TOT) = "%P''
 500 FP."U ZOCHT "%R" OHM"'''
 510 P."DIT KOMT AARDIG IN DE BUURT!"'
 520 BEEP 15,20;LINK#FFE3;G.30
 530fP.$12''"R1 = "RRM*F';?#E1=0
 540 P."R2 = "RRI*V'
 550 P."R3 = "RRU*Z'''
 560 FP."R(TOT) = "%Q''
 570 FP."U ZOCHT "%R" OHM"'''
 580 P."ZIT IK ER NIET VER NAAST!!"'
 590 BEEP 15,20;LINK#FFE3;G.30
 600t %O=%R-(%T*%R/100)
 610  %B=%R+(%T*%R/100)
 620  RETURN
 630 DATA 10,12,15,18,22,27,33,39,47,56,68,82,100
 640 END
```

Frens Nepschen

ACORN
COMPUTER CLUB
"BRABANT-OOST"

OP VELER VERZOEK HIER NOG ENKELE AANPASSINGEN VOOR DE JOYSTICK. OOK HIER WEER EERST DE SPELEN LADEN VAN TAPE, EN DAN DE VOLGENDE AANPASSINGEN IN TYPEN:

BREAKOUT

```
?#31DA=2
?#31DE=1
?#31E1=8
1700 [F] DOU.?0&1=0
4200     DOU.?0&1=0
```

SPACE ATTACK

```
530IF?0&2=0G=G-2
540IF?0&8=0G=G+2
560IF?0&1 R.
```

ASTROBIRDS

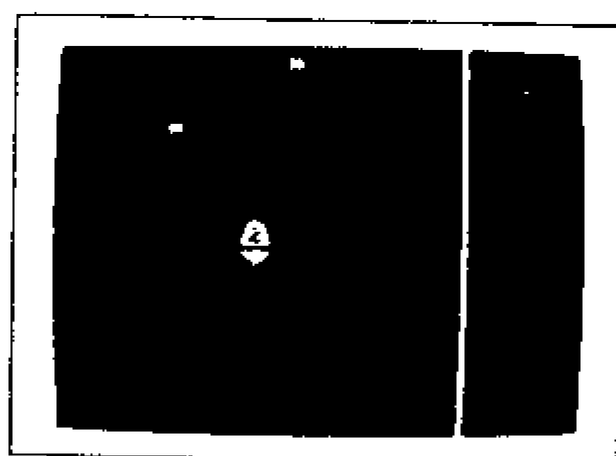
```
?#31DE=8
?#31F9=2
?#3256=1
?#3259=1
225IF?#A0=#13G.[L]
230IF?#A0<>#27G.220
```

CYLONATTACK

```
?#3369=16
?#3377=4
?#3389=8
?#339B=2
```

LUNAR LANDER

```
60IF?Q&15=0
75D=D+(?Q&8=0)*12-(?Q&2=0)*6
170F.N=0 TO 6;PLOT2,(RND%4),(RND%4)
   ;N.;LI.#FFE3;RUN
```



DODGEMS

```
?#3369=10
?#336B=10
?#3374=2
```

AMOEBA

```
200[L]F=?#B001:255
210IFF=8I=I%7+1;GOS.[P]
220IFF=16J=J%7+1;GOS.[P]
230                    (ALLEEN REGELNR.)
240IFF<>1G.[L]
```

MARTIANS

```
?#3813=2
```

JAN WIJNEN

ACORN-CALC

Dit artikel is niet bedoeld als een volledige beschrijving van het ACORN-CALC programma, maar alleen als een eenvoudige uitleg van de werking van het systeem. Voor een volledige beschrijving, zie het desbetreffende manual.

ACORN-CALC is te beschouwen als een heel groot kladpapier, wat ingedeeld is in vakjes.
Horizontaal 62 stuks: de letters A t/m BK.
Vertikaal 255 stuks: genummerd van 1 t/m 255.
Het vakje links boven in de hoek wordt dus aangeduid als A1 en helemaal rechtsonder als BK255.

Het beeldscherm is te beschouwen als een venster, waardoor we naar een klein gedeelte van ons kladpapier kunnen kijken.
De bovenste 3 regels van het scherm zijn gereserveerd voor de z.g. header.

Het gebruik: We kunnen de vakjes vullen met
1. Labels (etiket)
2. Getallen
3. Expressie's (formule)

Het programma bekijkt zelf, of het een label, getal of een formule betreft!
We kunnen de vakjes bereiken met behulp van de cursor, die op dezelfde wijze bestuurd kan worden als we gewend zijn met de normale cursor. We kunnen de cursor ook sturen met het commando /G (goto) met daarachter de coördinaten van het vakje waar we heen willen, b.v. /G C8
De schuine streep geeft aan, dat het een commando betreft.
Alles wat we intikken, verschijnt op de 3^e^ lijn van de header.
Wanneer we intikken: 1983, dan verschijnt dit op de derde lijn, en na het commando RETURN wordt 1983 geplaatst in het vakje waar de cursor zich op dat moment bevindt, stel dat dit vakje A1 is. 1983 verschijnt nu ook op de 2^e^ lijn; deze lijn geeft de aktuele inhoud van het vakje aangewezen door de cursor. Wanneer we nu de cursor 1 plaatsje laten zakken, dan verdwijnt 1983 van de 2^e^ lijn. We tikken nu in de formule A1+1 RETURN. Op de 2^e^ lijn verschijnt de aktuele inhoud van vakje A2, nu dus de formule A1+1 maar let op!, in vakje A2 komt nu niet de fomule maar het antwoord van de formule te staan, dus i.d.g. 1984. Verplaatsen we nu de cursor 1 stap omhoog, dan komt op de 2^e^ lijn weer 1983 te staan (de inhoud van A1). Wanneer we nu intikken 1984 RETURN, dan overschrijven we de oude inhoud en we zien tevens dat in vakje A2 automatisch het antwoord van de formule A1+1 verandert in 1985. Dit nu is de kracht van het calcprogramma. We kunnen ons kladpapier vullen met allerlei getallen en formules en het geheel overzichtelijk maken door het tussenvoegen van namen en verduidelijkingen (labels). Wanneer we één getal op ons kladpapier veranderen, dan rekent het programma alle antwoorden van de fomules opnieuw uit.
Het calcprogramma leent zich vanwege deze eigenschappen erg goed voor (financiële) planning, wanneer we bijvoorbeeld winstmarges willen uitrekenen die afhankelijk zijn van een aantal parameters zoals inflatie, rentestand, grondstoffenprijs etc.etc.
We kunnen nu bekijken wat de resultaten zijn bij een bepaalde inflatie. Veranderen we dit getal, dan wordt alles automatisch opnieuw doorgerekend.

Klaas de Raad

HIERBIJ EEN PROGRAMMA VAN EEN CONTINU DOORLOPENDE DEMONSTRATIE VAN DE KARAKTERSET, MET GEBRUIK VAN DE SHUTFUNCTIE UIT ACORNNIEUWS JRG.2 NR.1 BLADZ.49

```
10 REM DEMONSTRATIE KARAKTERSET
20 REM GEBRUIK JOSBOX OF TOOLKIT
30 REM TINY VERSCHUREN
40 G.H
50GP.$12
60 ?#E1=0
70 X=32
80CX=X+1
90 RESTORE
100AA=0
110 READ A
120 FOR P =1 TO 2;WAIT;NEXT
130 SHUT A;P.$X
140 IF A=71 T.GF
150 G.A
160FRESTORE
170EA=0
180 READ A
190 FOR R=1 TO 2;WAIT;NEXT
200 SHUT A;P.$32
210 IF A=71 T.GD
220 G.E

230DSHUT 450;P."*** U ZIET DE KARAKTERSET ***"'
240 IF X=#FF T.G.G
250 G.C
260 END
270 DATA 48,112,176,240,210,149,89,244,248,252,274,341,409
280 DATA 304,368,432,270,331,391,236,232,228,206,139,71
290HP=#2800;[LDY224;LDA(222),Y;EOR225;STA(222),Y;INX;LDA#42,X
300 ORA#33,X;BEQP+3;BRK;LDA#24,X;CMP@2;BCSP-5;ORA@128;STA223
310 LDA#15,X;AND@31;STA224;LDA#15,X;AND@224;STA222;RTS;]
320 !#21A=#2800;G.G
```

```
list
  10 f.n=0to16;p.'+-+-+-+-+ joy-muziek +-+-+-+-+';n.
  20 b=#b001
  30 do
  40 if?b=239;pl.g'8;g.a
  50 if?b=247;pl.d'8;g.a
  60 if?b=251;pl.g'8;g.a
  70 if?b=253;pl.d'8;g.a
  80 if?b=231;pl.e'#8;g.a
  90 if?b=243;pl.b'#8;g.a
 100 if?b=249;pl.e'#8;g.a
 110 if?b=237;pl.a'#8;g.a
 120 if?b%2=0:pl.r8;g.a
 130 pl.d8
 140au.0
rem auteur: hans otten.
rem dit programma maakt d.m.v. de joystick geluid
rem de hoogste toon krijgt u door de knuppel naar boven
rem te duwen.
rem de toon wordt met de klok mee steeds lager.
```

PROGRAMMA "sinusjes"
===================

Het is aan te raden om zoveel mogelijk de lijnnummers te behouden omdat er in regel 15 een onrechtstreekse GOSUB wordt berekend.

N.B. Lees "%" voor "¼".

```
l.
  1 rem door serge myngheer
  5 rem met toolbox
  7 p.$12''''''''"                   sinusjes"''''''''
  8 f.m=0 to 100;wait;n.
 10 dimaa(10),bb(10)
 15 f.p=75 to 145 s.10;gos.(p)
 25 if p=145;f.g=1 to 10
 30 if p=145;p=150;gos.(p)
 35 clear4;movea,b;f.i=0to c s.d;gos.(p+5);plotw,¼x,¼y;n.i
 45 f.l=8 to 120;beepl,1;l=-l;beepl,1;l=-l;n.l
 50 f.l=120 to 1 s.-1;beepl,1;n.l
 55 if p=150;p=145;n.g
 60 n.p
 65 p.$7$7$7$7$7;f.l=120 to 256;beepl,1;n.l
 67 p.$12''''''''''"                         einde"''''''''
 70 end
 75 a=0;b=0;c=14014;d=91;w=5;r.
 80 ¼x=(i+cos(i))/50;¼y=(i+sin(i))/50;r.
 85 a=0;b=0;c=9191;d=91;w=1;r.
 90 ¼x=(i+cos(i))+128;¼y=(i+sin(i))+96;¼x=¼x/50;¼y=¼y/50;r.
 95 a=0;b=0;c=9464;d=91;w=1;r.
100 ¼x=(2+(i+cos(i))+128)/50;¼y=(2+(i+sin(i))+96)/50;r.
105 a=0;b=0;c=2600;d=20;w=1;r.
110 ¼x=(2+(i+cos(i))+128)/50;¼y=(2+(i+sin(i))+96)/50;r.
115 a=0;b=0;c=9752;d=90;w=1;r.
120 ¼x=(2+(i+cos(i))+128)/50;¼y=(2+(i+sin(i))+96)/50;r.
125 a=0;b=0;c=9480;d=30;w=5;r.
130 ¼x=(2+(i+cos(i))+128)/50;¼y=(2+(i+sin(i))+96)/50;r.
135 a=128;b=96;c=504;d=3;w=6;r.
140 ¼x=2+i+cos(i)+1280;¼y=2+i+sin(i)+960;¼x=¼x/10;¼y=¼y/10;r.
145 restore;f.g=1 to 10;read q,r;aa(g)=r;bb(g)=q;n.g;r.
150 a=128;b=96;c=aa(g);d=bb(g);w=5;r.
155 ¼x=2+i+cos(i)+1280;¼y=2+i+sin(i)+960;¼x=¼x/10;¼y=¼y/10;r.
160 data4,512,6,690,7,728,8,816,9,1449,10,1230,11,770,5,975
165 data2,504,3,504
170 rem regels   7 t/m  10: initialisatie
175 rem regels  15 t/m  35: hoofdprogramma
180 rem regels  45 t/m  70: eindgeluiden + definitief einde
185 rem regels  75 t/m 165: formules + gegevens voor tekeningen
190 rem regels  75 t/m  80: tekening nr.1
195 rem regels  85 t/m  90: tekening nr.2
200 rem regels  95 t/m 100: tekening nr.3
205 rem regels 105 t/m 110: tekening nr.4
210 rem regels 115 t/m 120: tekening nr.5
215 rem regels 125 t/m 130: tekening nr.6
220 rem regels 135 t/m 140: tekening nr.7
225 rem regels 145 t/m 165: tekening nr.8 t/m 18
```

TOELICHTING BIJ DEMONSTRATIE-PROGRAMMA "sinusjes"
===

Bij het napluizen van de verschillende interessesferen die (elk van ons) U heeft ingevuld op het INFO-formulier, stel ik vast dat velen onder U grote interesse tonen voor GRAFICS.
Daarom dacht ik dat het interessant zou kunnen zijn om navolgend klein programmaatje te laten afdrukken in ons clubblad.
Het is geen "gecopieerd" programma. Ik heb alle gegevens per toeval gevonden tijdens eenpoging om een tekenprogramma voor de HP-85 om te schrijven naar ATOM-basic.
Het programmaatje runt met de TOOL(box). Indien U een andere TOOL-(box, kit, ...) hebt, kan U het misschien aanpassen door rekening te houden met volgende gegevens :

- BEEP x,y geeft een toon van toonhoogte x en duur y
- RESTORE is verplicht vóór READ en DATA
- READ x plaatst de waarde van de DATA in x
- DATA x,x,x,x,... zijn de data voor READ

Indien U echter geen TOOL (box, kit,...) hebt moet U de lijnen 45, 5Ø en 6Ø vervangen door :

```
50 FOR L = 0  TO 200 ; WAIT ; NEXT L
60 PRINT $ 7 $  7 $ 7
```

Daarenboven moet U ook de lijnen 145, 16Ø en 165 vervangen door :

```
145 AA(I) = 512; BB(I)=4;AA(2)=690;BB(2)=6;enz..tot;AA(I0)=
504;
     BB(I0)= 3 (deze getallen zijn af te leiden uit de data)
147 RETURN
```

Het programma neemt ongeveer 5 minuten om helemaal te zijn uit-gerund.
Ik hoop dat elk van U ookeens een klein programma instuurt.

Vriendelijke groeten,

SERGE MYNGHEER

TIPS uit het Noord'n.

- Als we met het kommando STEP uit de Josbox een machinetaal programma doorlopen dat gebruik maakt van de VIA (6522) gaan er vaak dingen fout. Dit is bijvoorbeeld het geval met de printerroutine (≠FEFB - ≠FF3E). Oppassen dus.

- Het spelletje RAKETBASIS geeft problemen bij het opnieuw saven op de band, ikzelf heb het als volgt opgelost:

  Eerst het (Telec-) origineel op 300 baud laden met *RUN"RAKETBASIS", daarna:

  ```
  break
  OLD
  regel 5 verwijderen
  ?18=≠8C
  END
  regels 5,10,15,20 en 30 verwijderen
  regel 600 veranderen in:  600 LINK≠93D3;G.600
  ```

  Vervolgens kan het programma opnieuw op de band worden gezet (eventueel op 1200 baud) door eerst in te typen:

  ```
  *SAVE"RAKETBASIS"2900 3900
  ```

  Hierna moet de band iets worden teruggespoeld. Vervolgens *CAT en wachten tot het blok met startadres 3700 wordt bereikt. Dan (op het gehoor)wachten tot dit blok is afgelopen en de recorder zo snel mogelijk uitschakelen (voordat het volgende blok begint). Daarna intypen:

  ```
  *SAVE"RAKETBASIS"8C00 9800
  ```

  Het laatste blok van het eerste deel (3800-3900, dummy-blok) wordt dan weer overschreven met het eerste blok van het tweede deel.
  De komputer beschouwt nu de twee delen als één geheel (ze moeten wel dezelfde naam hebben).

  Het programma kan nu weer worden geladen en gerunt met:

  ```
  *FLOAD"RAKETBASIS"
  break
  OLD
  RUN
  ```

  Hier wordt *FLOAD gebruikt omdat de bloknummers niet meer kloppen, het tweede deel begint weer met bloknummer 0. *FLOAD trekt zich van deze bloknummers niets aan.

  Deze methode kan algemeen worden toegepast om twee programma's in verschillende tekstruimten als één geheel op de band te zetten.

- Het is niet mogelijk om in een expressie met twee strings de operators <>,>,<,>= en <= te gebruiken, alleen = is toegestaan.
  IF $E<>$L THEN .... mag dus niet.

- Sommige programma's zoals PEEKO en DATABASE werken niet als ze in de 16k-kaart worden gezet.
  Bij beide programma's hoort nog een machinetaal-gedeelte dat na het Basic-programma komt. Als het programma in de 16k-kaart (of op de band) wordt gezet moet dit stuk machinetaal natuurlijk worden meegenomen. Het eindadres van het gehele programma kunnen we aflezen bij *CAT (startadres laatste blok + aantal bytes laatste blok). Als het programma in de 16k-kaart is gezet moet het eerst weer geheel (incl. machinetaal) orden gekopieerd naar de oorspronkelijke lokatie (bijv. #2900) voordat ermee gewerkt kan worden.

- Als we met het statement RELOC (Josbox) delen uit rom's of eprom's gaan verplaatsen kan het fout gaan met de eventueel aanwezige tabellen. Staat namelijk ergens in een tabel toevallig de hex-kode voor een instruktie die absolute adressering gebruikt (bijv. JSR, kode #20), dan worden de twee hierop volgende bytes in de tabel veranderd. Dit gebeurt echter alleen als het 'adres' gevormt door de twee bytes binnen het verplaatste gedeelte ligt (in de praktijk komt dit niet zo vaak voor). Tabellen kunnen dus het beste met COPY worden verplaatst.

- De transistor op de schakelkaart moet andersom, maar dat weet dacht ik iedere schakelkaart-bezitter al.

- De kombinatie van de 16k-kaart en de schakelkaart op de bus geeft problemen. Aan dit onderwerp kan een hele thema-avond besteed worden en ik zal er daarom ook niet verder op in gaan. In ieder geval is het goed om de CS-lijn van het gestapelde lage geheugen (TC6, pen 7) te verbinden met IC5, pen 1 (pootje uit socket buigen) zodat bij adressering van dit geheugen de busbuffers (IC 2,3,4) niet worden opengestuurd.

Johan de Goede

Naschrift redactie: Algemene problemen bij de combinatie schakelkaart+geheugenkaart zijn bij dhr Borghaerts niet bekend. Het "binnen de bus" houden van het gestapelde lage geheugen is sowieso aan te bevelen en bij gebruik van de 16K-kaart zelfs noodzakelijk!

Tenslotte nog een tipje van uw redacteur, uit de Randstad dus.

Het betreft het DATA en READ statement van de Josbox.

Als u tegen uw Atom zegt: READ $A , dus met een spatie ertussen, dan kunt u een string inlezen van maximaal 47 karakters.

Zegt u evenwel: READ$A , dus zonder spatie ertussen, dan kunt u maximaal 48 karakters inlezen. Bovendien wordt een komma of een puntkomma in de string door het READ-statement beschouwd als het einde van de string. Knap vervelend, als je daar niet op bedacht bent. Het is maar een weet.

Ton Otten.

TIPS uit het Noord'n (vervolg).

Laat ik alvast eens beginnen met te vertellen hoe commando's gelezen worden.

Ik zal de in aanmerking komende geheugenplaatsen even noemen:

≠ ØØØØ Hier staat het ERROR nummer.
≠ ØØØ1
≠ ØØØ2 In deze geheugen plaatsen staat het regelnummer.
≠ ØØØ3 Hier staat de karakter pointer.
≠ ØØØ5
≠ ØØØ6 Hier staat de statement pointer.
≠ ØØ1Ø
≠ ØØ11 Hier staat de error vector (normaal ≠ C9E7)

<u>Wat gebeurt er nu precies</u> ?

- Als op het beeldscherm verschenen is : ERROR 174 en u toetst in P.?Ø dan verschijnt het error nummer weer. In dit geval dus 174.
- Wat doet de Atom eigenlijk met ≠ ØØØ1 t/m ≠ ØØØ6 ?
  Stel dat je het volgende stukje programma hebt:
  1ØØ A=1; RETURN; X=1ØØ
  11Ø WAIT; X=X+4
  Eerst wordt het regelnummer ingelezen in ≠ ØØØ1 en ≠ ØØØ2 (1ØØ dus), dan wordt de inhoud van ≠ ØØØ3 met 1 verhoogd nadat de spatie gelezen is.
  dan wordt de 'A' gelezen en de inhoud van ≠ ØØØ3 wordt 2.
  dan wordt de '=' gelezen en de inhoud van ≠ ØØØ3 wordt 3.
  dan wordt de '1' gelezen en de inhoud van ≠ ØØØ3 wordt 4.
  dan wordt de ';' gelezen en de inhoud van ≠ ØØØ3 wordt weer Ø, maar de inhoud van ≠ ØØØ5 en ≠ ØØØ6 wordt nu '1'(highbyte = Ø, lowbyte = 1).
  Hierna begint het hele spelletje opnieuw.

- Als er iets gevonden wordt wat niet klopt volgens de atom dan wordt gesprongen naar waar de error vector heen wijst.
  Dus als je de error vector bijv. verzet naar een plaats waar een ander programma begint, dan wordt bij een error daarheen gesprongen.

<u>Wat gebeurt er bij BREAK</u>??

- 1 Hoge byte van adres stack (≠ C55C)
  2 Lage byte + 2 van adres op stack)
  3 Processorstatus op de stack
  4 JMP (≠ FFFE)
  is ≠ FFB2 voor de atom
  FFB2 Eindigt met JMP(≠ Ø2Ø2), normaal staat op ≠ 2Ø2, ≠ 2Ø3 de waarde ≠ C9D8.

  <u>Wijzigen is handig</u> voor extra statements of nieuwe functies of betere error meldingen.
  Een groot nadeel is dat de Acorn Atom niet met zijn digitale vingers van ≠ 2Ø2, ≠2Ø3 afblijft maar hem terug zet op ≠ C9D8.

Wim Schreuder

```
LIST
   10REM /// COPY /// V1
   20REM (C) K.V.HOUTEN 1983
   30P.$12" *** tape to tape ***"'
   40P=#2300;P.$21![JSR#FFE6;STA#323;RTS;]:P.$6
   50P."FAST OR SLOW ? (F/S)";LI.#2800;P.'
   60FC.;IF B&#FF=#53 SC.
   70P."FILENAME ";A=#3A;LI.#CD0B
   80!#C9=#82000140;?#CD=#FF
   90LI.#F969;P.$7'" LOADED FROM TAPE "''
  100P=((?#D5-?#D9)*#100+?#D4)
  110Q=(?#D5*#100)+?#D8
  120R=(?#D7*#100)+?#D6
  130P."FILEDATA:"'
  140P." STARTADRESS #",&P'
  150P." ENDADRESS   #",&Q'
  160P." EXECADRESS  #",&R'
  170P.'" (PRESS SPACE) ";LI.#FFE3
  180!#CB=P;!#CD=R;!#CF=#8200;!#D1=(#8200+(Q-P)+1)
  190P.$12" *** tape to tape ***"'
  200P."SAVING FILE: "$#140''
  210P."FAST OR SLOW ? (F/S)";LI.#2800;P.'
  220FC.;IF B&#FF=#53 SC.
  230LI.#FAE0
  240P.$7'" SAVED TO TAPE"'';E.
```

Noot van de redactie:

Een bijzonder handig programma'tje, dat de hele dialoog, nodig om het te copiëren programma te laden en weg te schrijven, begeleidt door alle benodigde gegevens te vragen. Het besturen van het COS doet het programma'tje zelf, inclusief het instellen van FCOS of SCOS (Josbox nodig!). Het te copiëren programma wordt geladen in het video-RAM, ongeacht het startadres of executie-adres op de band.

Het startadres en het executieadres van de copie kunt u zelf opgeven, dus RELOCEREN kan dit programma ook!!!